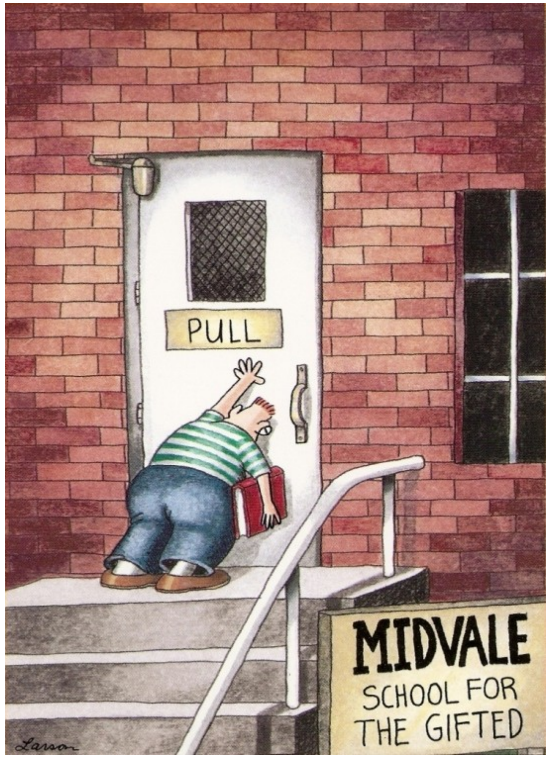
“Cartoonist Gary Larson Taught Me How To Build Infinitely Scalable High Performance Systems”

Brad Whitehead

Chief Scientist

Formularity



SouthEast LinuxFest

SELF 2023

10 June 2023

Who is Brad Whitehead ?!?!

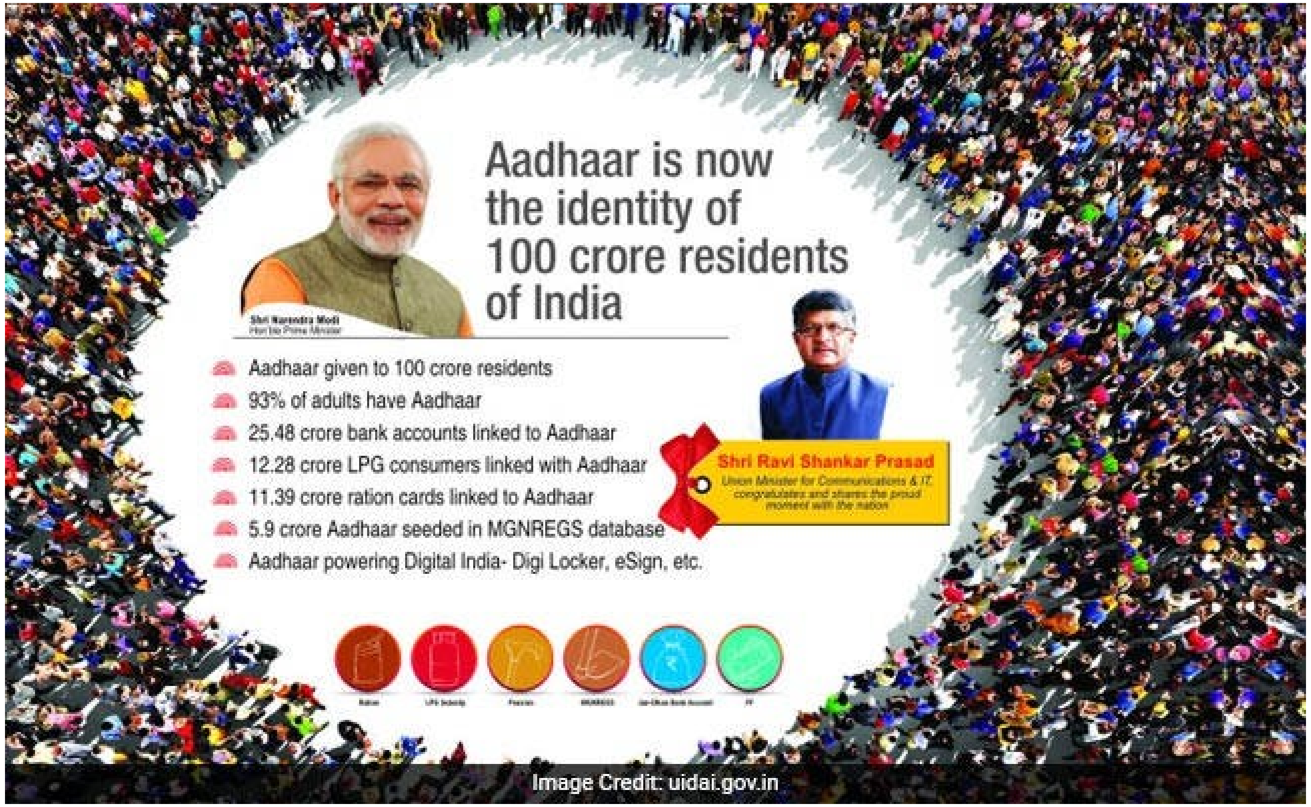
- Former Partner and Master Technology Architect with Accenture
  - National Scale Biometric Identification and Border Management Systems
    - US VISIT (one of CBP’s Biometric Border Control programs)
    - DHS Transportation Worker’s Identification Card (TWIC)
    - TSA PreCheck
    - Republic of India’s Aadhaar Program

- Presently Co-Founder and Chief Scientist of Formularity
  - Secure Electronic Enrollment Forms for the Government, Healthcare, Finance, and Legal Industries
  - We Offer a Hosted Solution, in Addition to Our Primary, On-Premise Products
  - We Take the Security of Our Client’s Information VERY Seriously
  - We Constantly Investigate and Explore New Advances in Information Security and Assurance
  - Currently Open Sourcing Aspects of the Aadhaar Biometric Processing Technology

UIDAI

Unique Identification Authority of India
Planning Commission, Government of India

AADHAAR

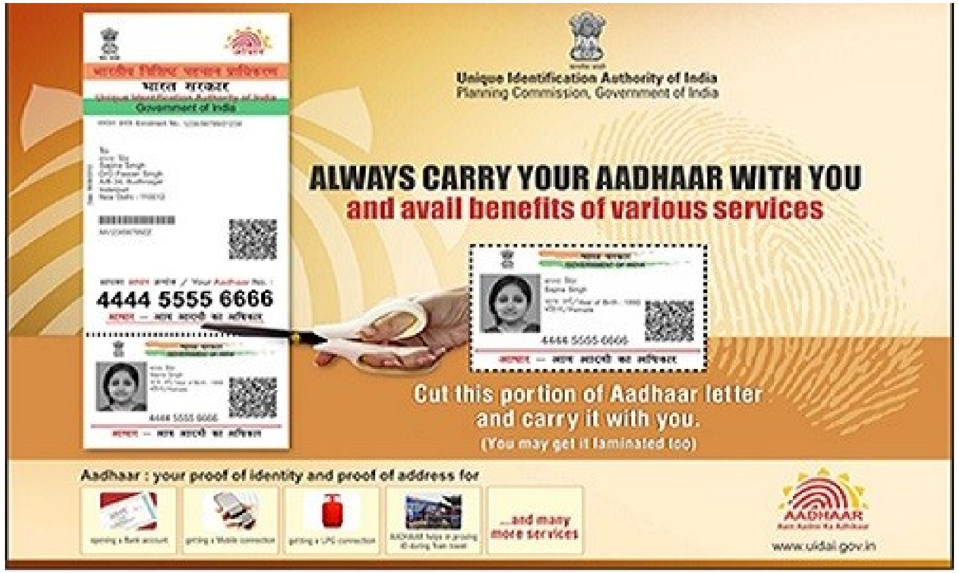


Aadhaar is now the identity of 100 crore residents of India

- Aadhaar given to 100 crore residents
- 93% of adults have Aadhaar
- 25.48 crore bank accounts linked to Aadhaar
- 12.28 crore LPG consumers linked with Aadhaar
- 11.39 crore ration cards linked to Aadhaar
- 5.9 crore Aadhaar seeded in MGNREGS database
- Aadhaar powering Digital India- Digi Locker, eSign, etc.

Shri Ravi Shankar Prasad
Union Minister for Communications & IT, congratulates and shares the proud moment with the nation

Image Credit: uidai.gov.in



Unique Identification Authority of India
Planning Commission, Government of India

ALWAYS CARRY YOUR AADHAAR WITH YOU
and avail benefits of various services

Cut this portion of Aadhaar letter and carry it with you.
(You may get it laminated too)

Aadhaar : your proof of identity and proof of address for

opening a Bank account
getting a Mobile connection
getting a LPG connection
AADHAAR helps in proving ID during Train travel
...and many more services
AADHAAR
Aam Aadmi Ka Adhikaar

www.uidai.gov.in

AADHAAR – “The Last Day”

Comparing 1 Million New Enrollees…

…Against 1 Billion Existing Enrollees

$1x10^6$ X $1x10^9$ = $1x10^{15}$

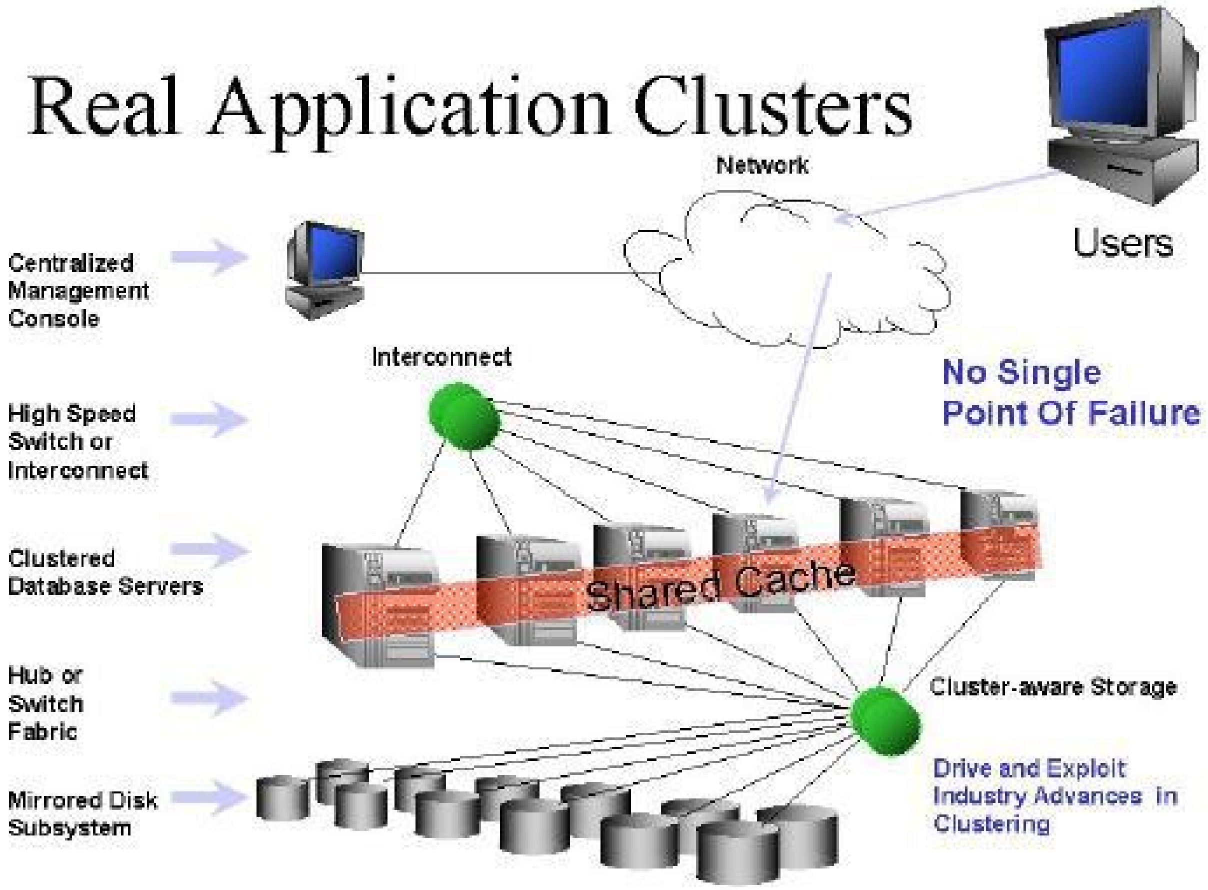
1 Quadrillion Comparisons in a day!

If you are using a database for workflow, that’s:

11 Billion Transactions/Second!

(It’s Even More If You Individually Log Each of 10 Fingers, 2 Irises, and a Face)

Real Application Clusters



Centralized Management Console

High Speed Switch or Interconnect

Clustered Database Servers

Hub or Switch Fabric

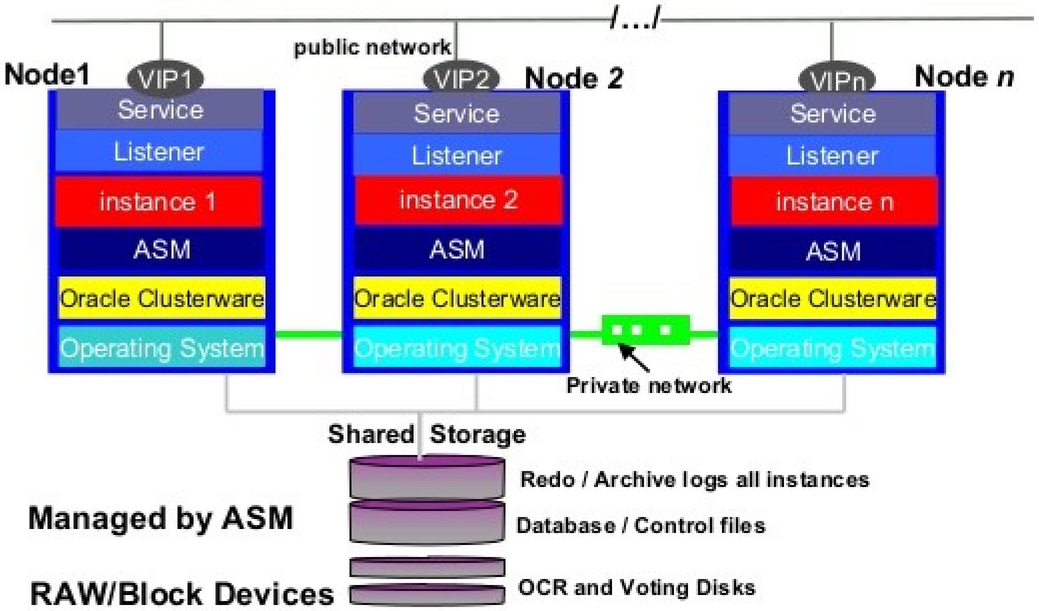
Mirrored Disk Subsystem

No Single Point Of Failure

Cluster-aware Storage

Drive and Exploit Industry Advances in Clustering

Oracle RAC Architecture



public network

Node1

VIP1

Service

Listener

instance 1

ASM

Oracle Clusterware

Operating System

Node 2

VIP2

Service

Listener

instance 2

ASM

Oracle Clusterware

Operating System

Node *n*

VIPn

Service

Listener

instance n

ASM

Oracle Clusterware

Operating System

Private network

Shared Storage

Redo / Archive logs all instances

Database / Control files

OCR and Voting Disks

Managed by ASM

RAW/Block Devices

What Customers Need to Know About RAC

"Perfect" RAC scalability occurs when:

- Data structures are regular and evenly divisible
- Data can be partitioned equally

Then:

- RAC will scale very well
- incremental 0.8 – 0.9
- shown to scale to 16 nodes and beyond

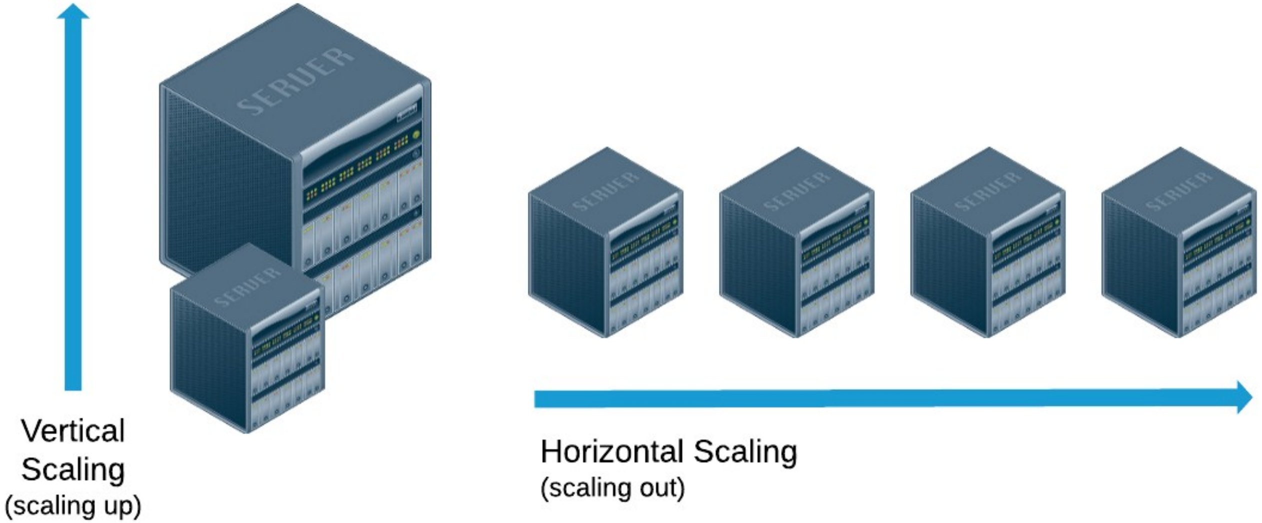
"Real" RAC scalability occurs when:

- Data structures are irregular and not evenly divisible
- Data can't be partitioned equally

Then:

- RAC will scale a lot less well
- 2nd node: incremental 0.6 – 0.7
- 3rd node: incremental 0.5 – 0.6
- 4th node: incremental 0.4 – 0.5
- 5th node: incremental zero - to - negative

Scaling a System (Ideally)



The Root of the Problem: Shared Resources

What Goes Wrong:

- Corruption
- Race Conditions
- Deadly Embrace
- Blocked Processes and Threads

Solutions(?):

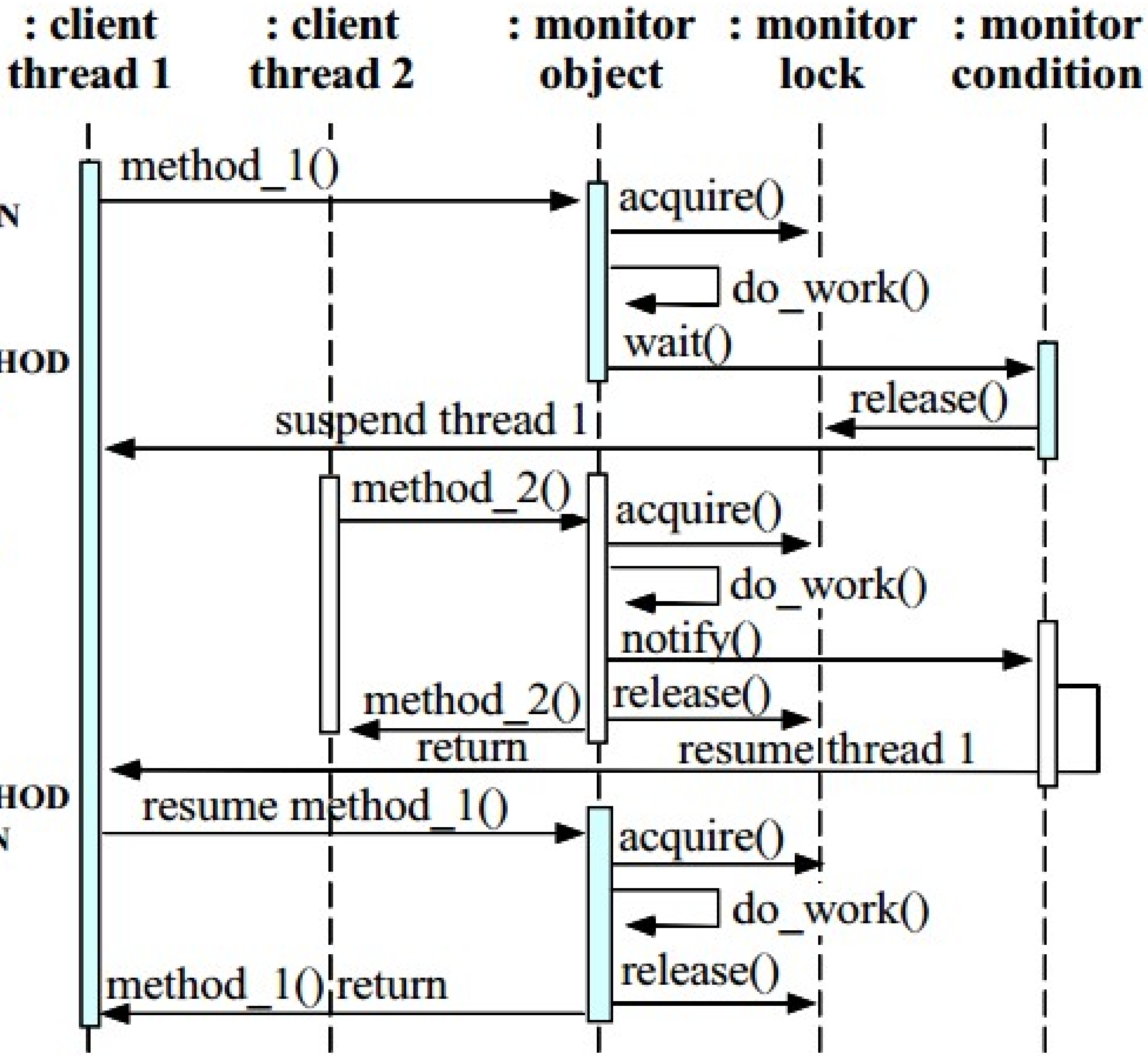
- Locks
- Mutexes
- Semaphores
- Latches
- Monitors

SYNCHRONIZED METHOD INVOCATION & SERIALIZATION

SYNCHRONIZED METHOD THREAD SUSPENSION

METHOD CONDITION NOTIFICATION

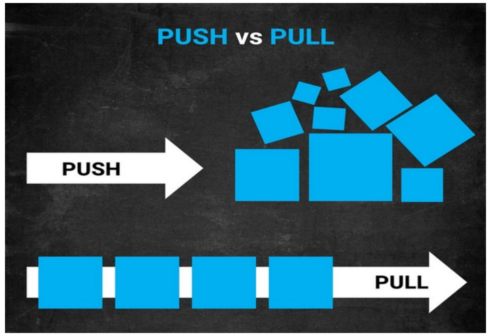
SYNCHRONIZED METHOD THREAD RESUMPTION



The Four Ingredients in the "Secret Sauce" for Developing Infinitely Scaling High Performance Systems

1) (Actors)

2) P's and Queues

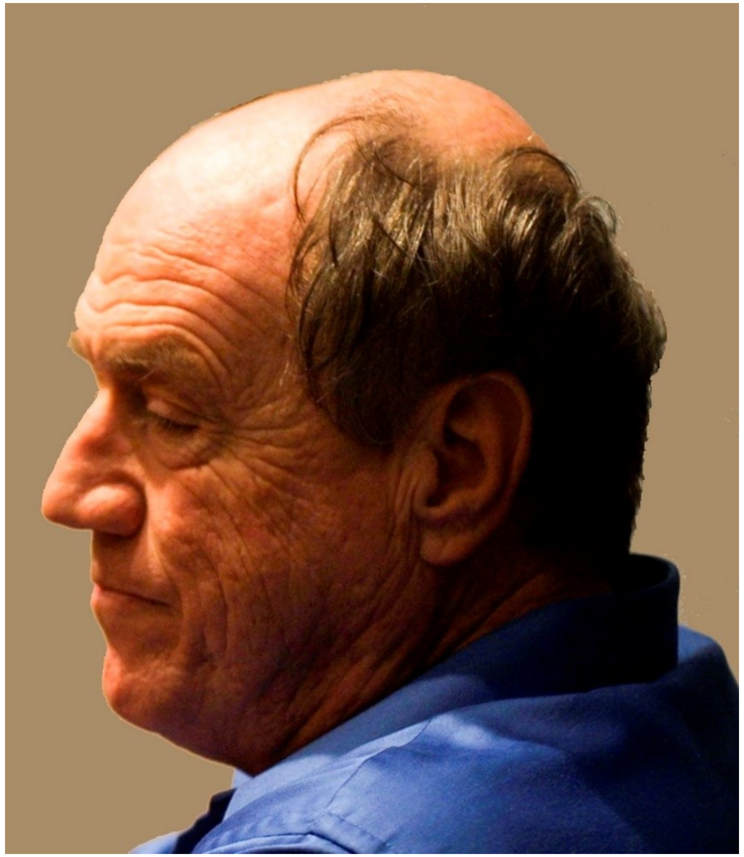
3) 

4)

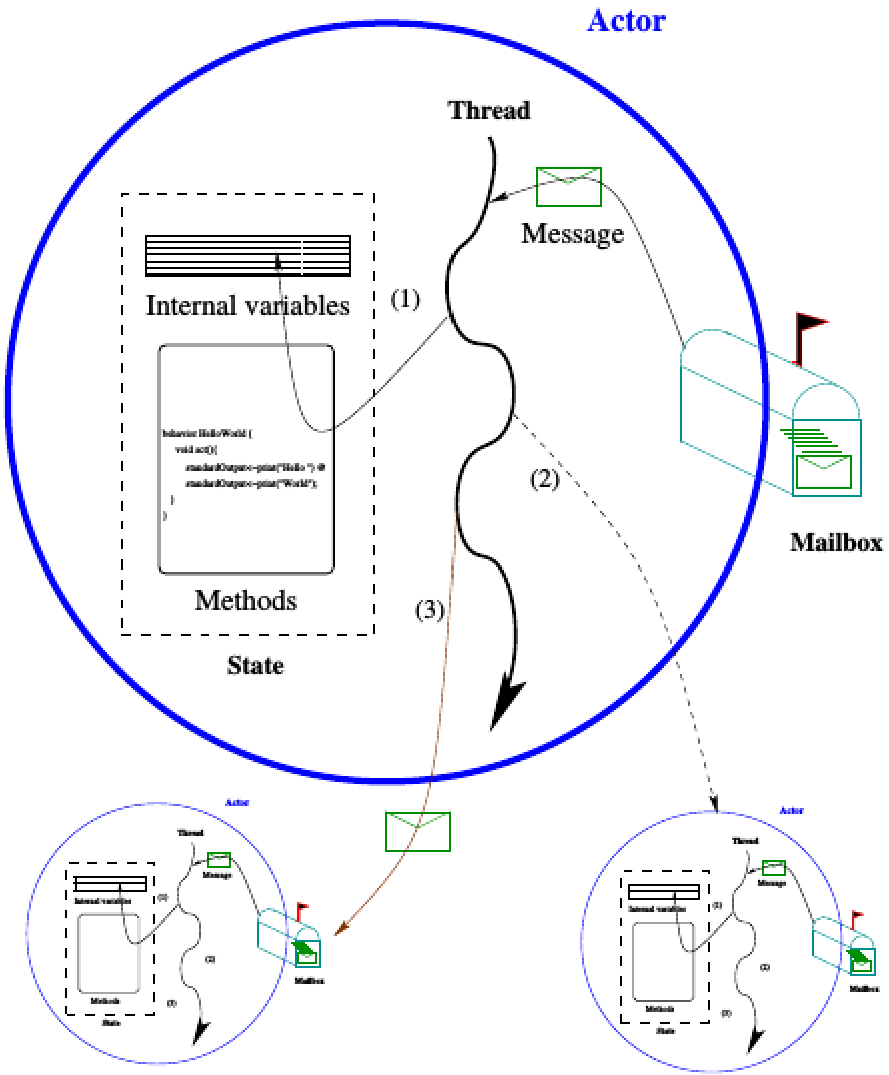
(Unikernels in Light VMs)

The Four Ingredients in the "Secret Sauce" for Developing Infinitely Scaling High Performance Systems

1) (Actors)



Carl Hewitt



```
behavior HelloWorld {
    void act(){
        standardOutput<-print("Hello ") @
        standardOutput<-print("World");
    }
}
```

Figure 1.2: In response to a message, an actor can: (1) modify its local state, (2) create new actors, and/or (3) send messages to acquaintances.

The Actor Model

Actors...

1) ...make local decisions based on its finite state table (Script or Program);

2) ...create new Actors;

3) ...send out messages;

4) ...adjust its internal state to a new state (which will reflect on how it acts to future messages)

5) ...communicate through messages in mailboxes

Core Network Products

carritech
telecommunications

Ericsson AXD Series

AXD 301	Product Name	Product Type	Key Features & Uses	Pricing
	Ericsson AXD 301	Soft-switch solution for telephony over IP.	Ericsson AXD 301 supports IP, MPLS, ATM, Frame Relay, circuit emulation and voice services and any combination of services can be handled simultaneously.	**Click here to request a quote for the complete system**
	Description			
	It is also one of the components in the Engine Integral Network (EIN) which is a Soft-switch solution for Telephony over IP. AXD 301 is also the Telephony Access Gateway in the Access Network Consolidation Solution that offers modernisation of the narrowband access network combined with full BB coverage with high capacity bandwidth. AXD 301 can even used as a Core Switch in the backbone network. The system scales from 10 Gbit/s, in one sub-rack, up to 160 Gbit/s.			

AXD 305	Product Name	Product Type	Key Features & Uses	Pricing
	Ericsson AXD 305	Multiservice Switch	Ericsson AXD 305 which hosts up to 160 E1/DS1 ports in a standard 19-inch rack, is a compact version of the AXD 301 Multi-Service Switch.	**Click here to request a quote for the complete system**
	Description			
	Ericsson AXD 305 which hosts up to 160 E1/DS1 ports in a standard 19-inch rack, is a compact version of the AXD 301 Multi-Service Switch. There is compatibility between the two AXD products; software, circuit boards, power supply equipment, and CPU modules are interchangeable. Both the AXD 301 and 305 can be used as an MGW (Media Gateway) in the EIN.			

For more products and information regarding Carritech and the wider telecommunications industry, visit our website at www.carritech.com or follow us on twitter @carritech

Actors = Microservices

Actor-Microservice Characteristics

1) Each Actor is small, accomplishing a single function. The finite state table (or its programming equivalent) is bounded and traceable. This means that the “correctness” of each Actor type is more easily established.

2) Each Actor type can be better tested because of its encapsulation nature and message passing framework.

3) Actors can be easily improved or re-factored without affecting the rest of the application.

4) Actors can be easily connected together to respond to changes in requirements or to provide new services.

5) The only way one Actor can affect another Actor is through explicit messaging. This makes it much easier to see and avoid concurrency problems.

Actor-Microservice Characteristics – Part Deux

6) Actors can create new Actors, to replace failed Actors or to increase the processing power at any step in the overall process

7) Actors can encapsulate common resources and act as service or resource provider

8) Multiple Actors can increase availability and/or persistence

9) If each Actor is run on a separate server, the Actor code may fit entirely within the processor cache, greatly accelerating the execution speed

10) If each Actor is run on a separate server, geographical diversity for improved resiliency is possible.

Actors – “Good Stuff!

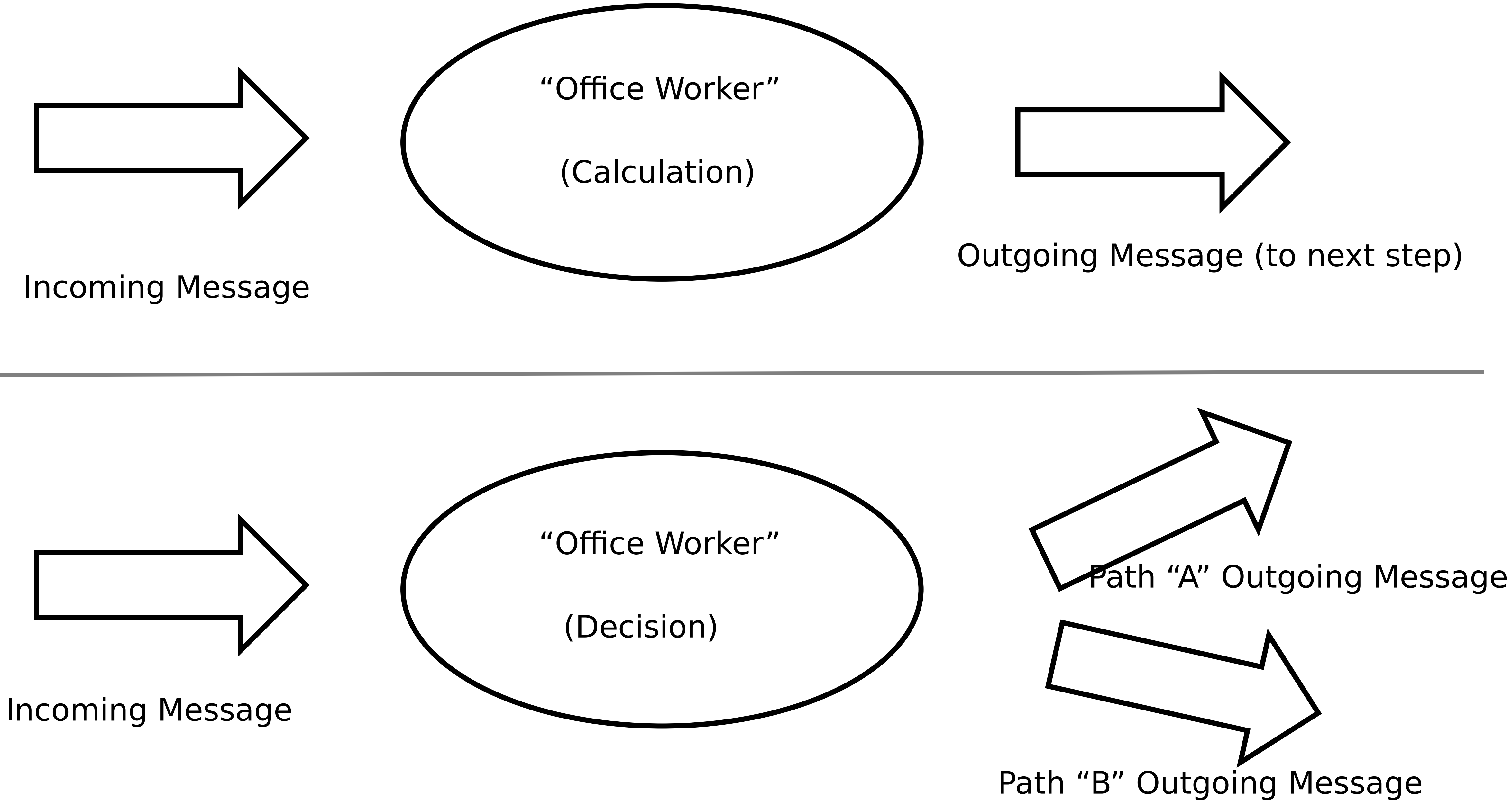
Business Process Actor Patterns

1. The Office Worker

2. The Supply Clerk

3. The Supervisor

Actor Type: “Office Worker”



“Office Worker”

(Calculation)

Incoming Message

Outgoing Message (to next step)

“Office Worker”

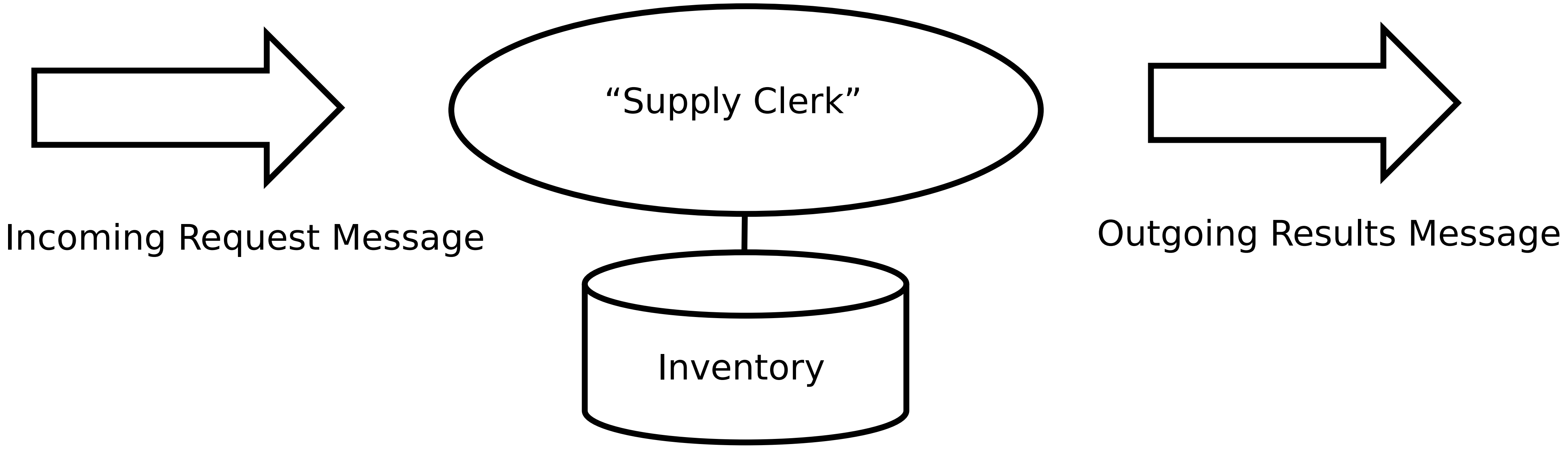
(Decision)

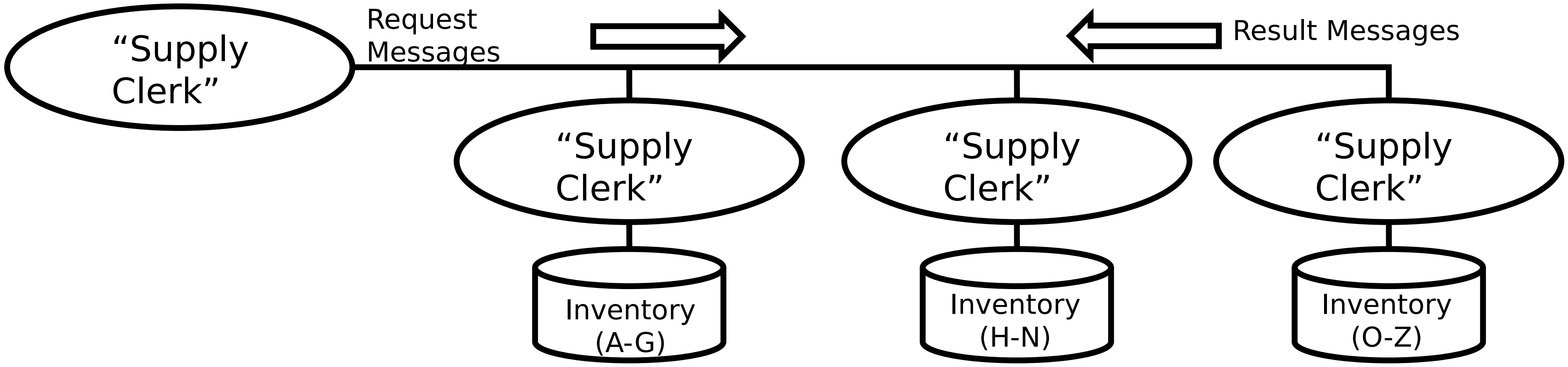
Incoming Message

Path “A” Outgoing Message

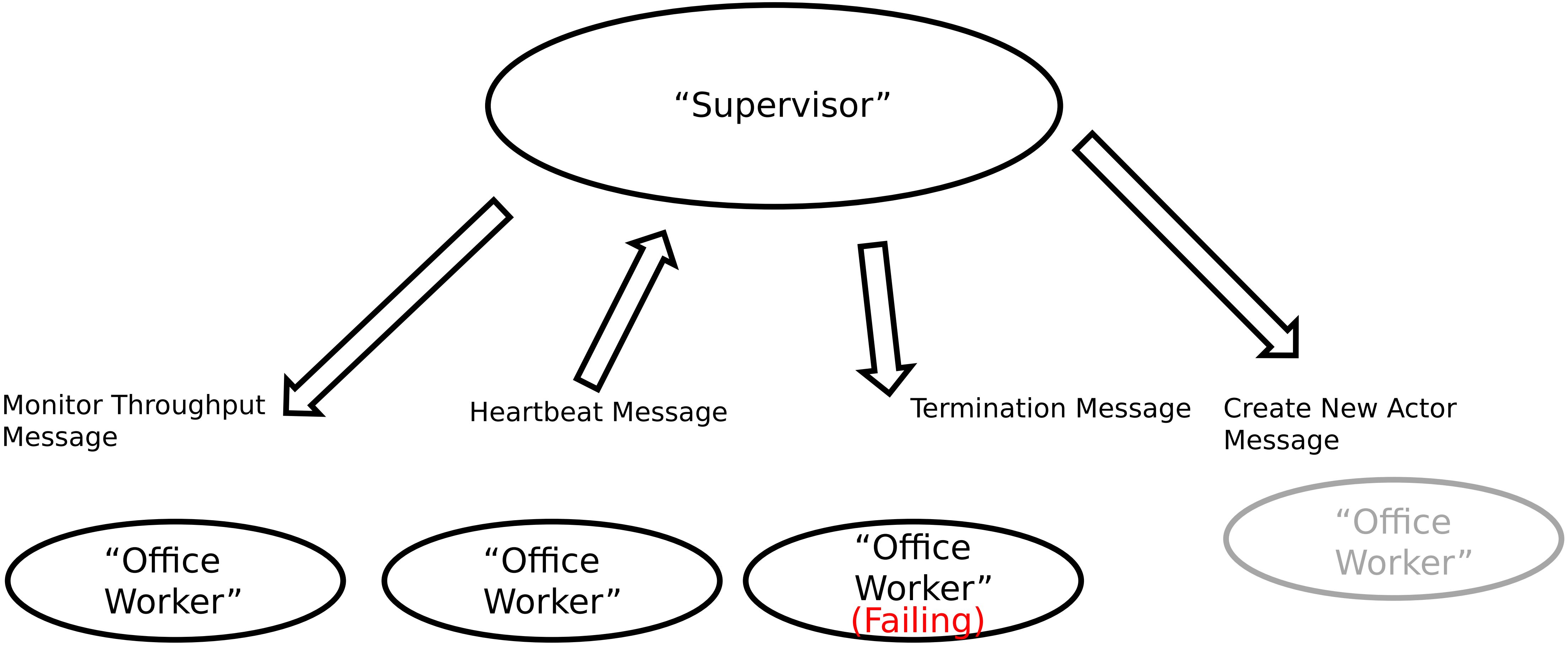
Path “B” Outgoing Message

Actor Type: “Supply Clerk”





Actor Type: “Supervisor”

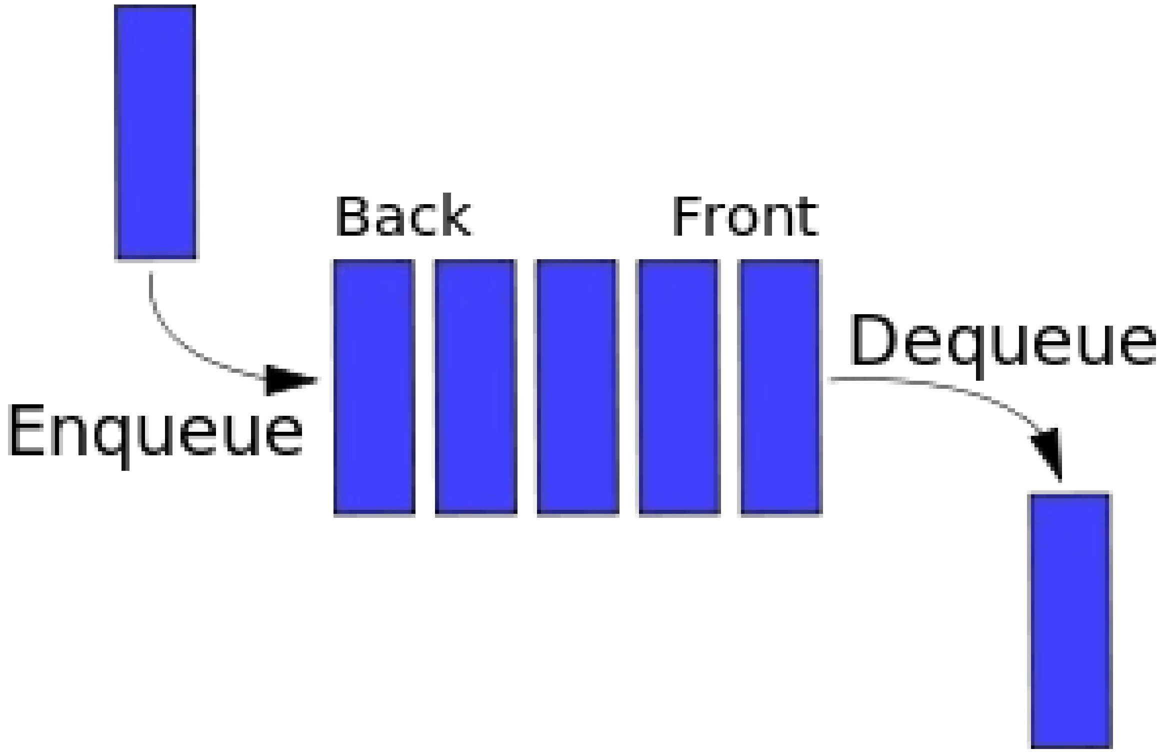


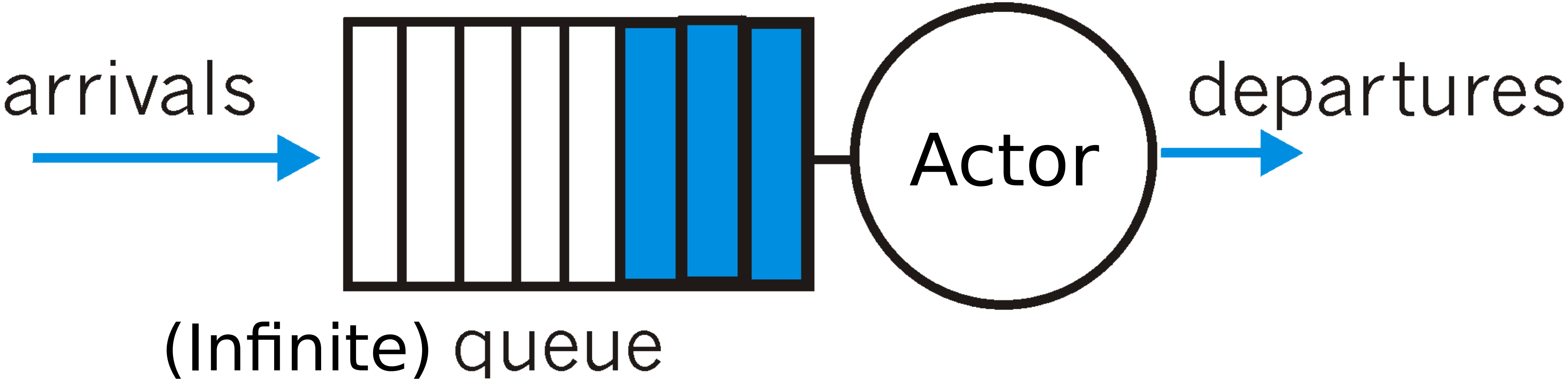
Responsible for the Health, Configuration and Performance of Their Actors

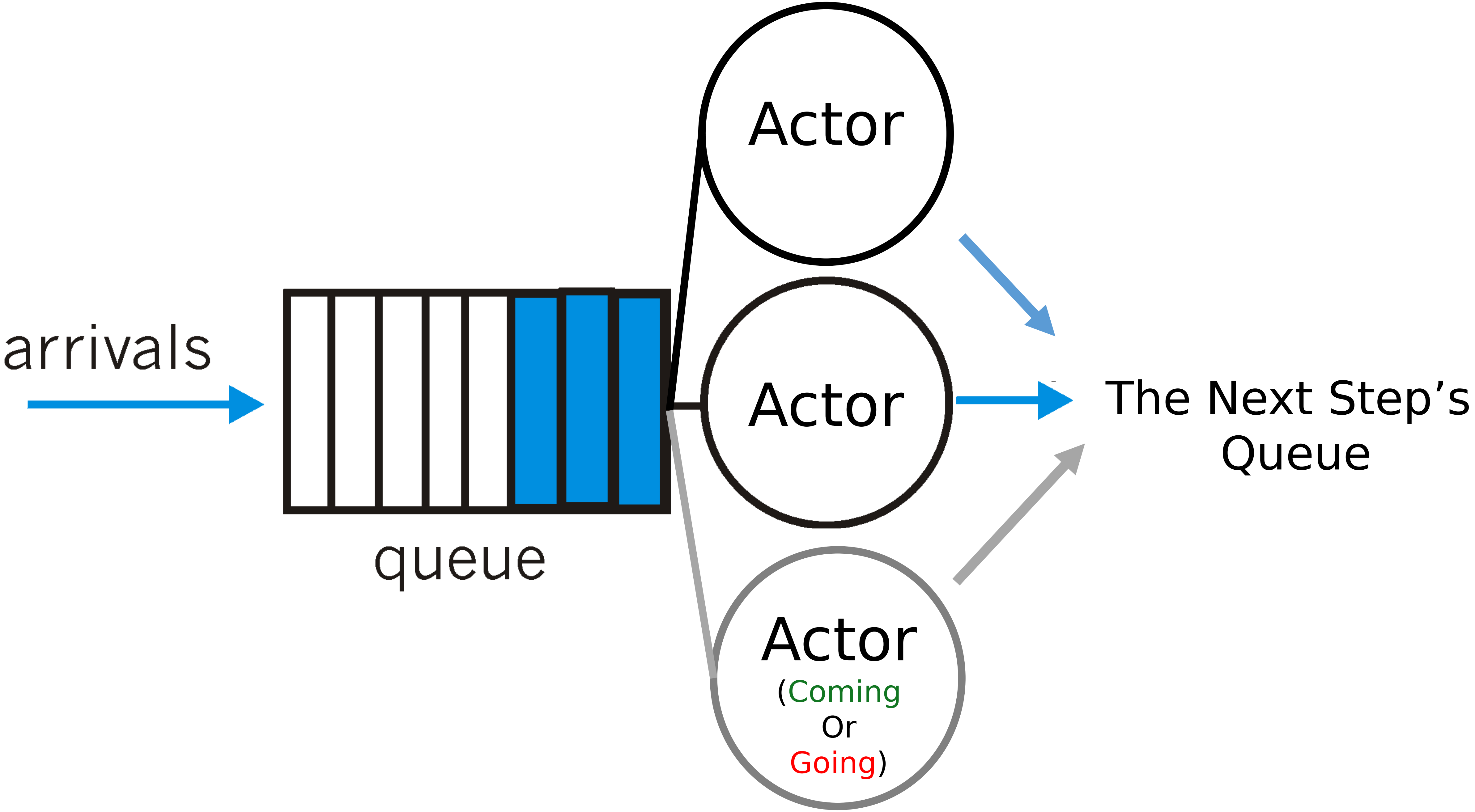
The Four Ingredients
in the
“Secret Sauce”
for
Developing Infinitely Scaling High Performance Systems

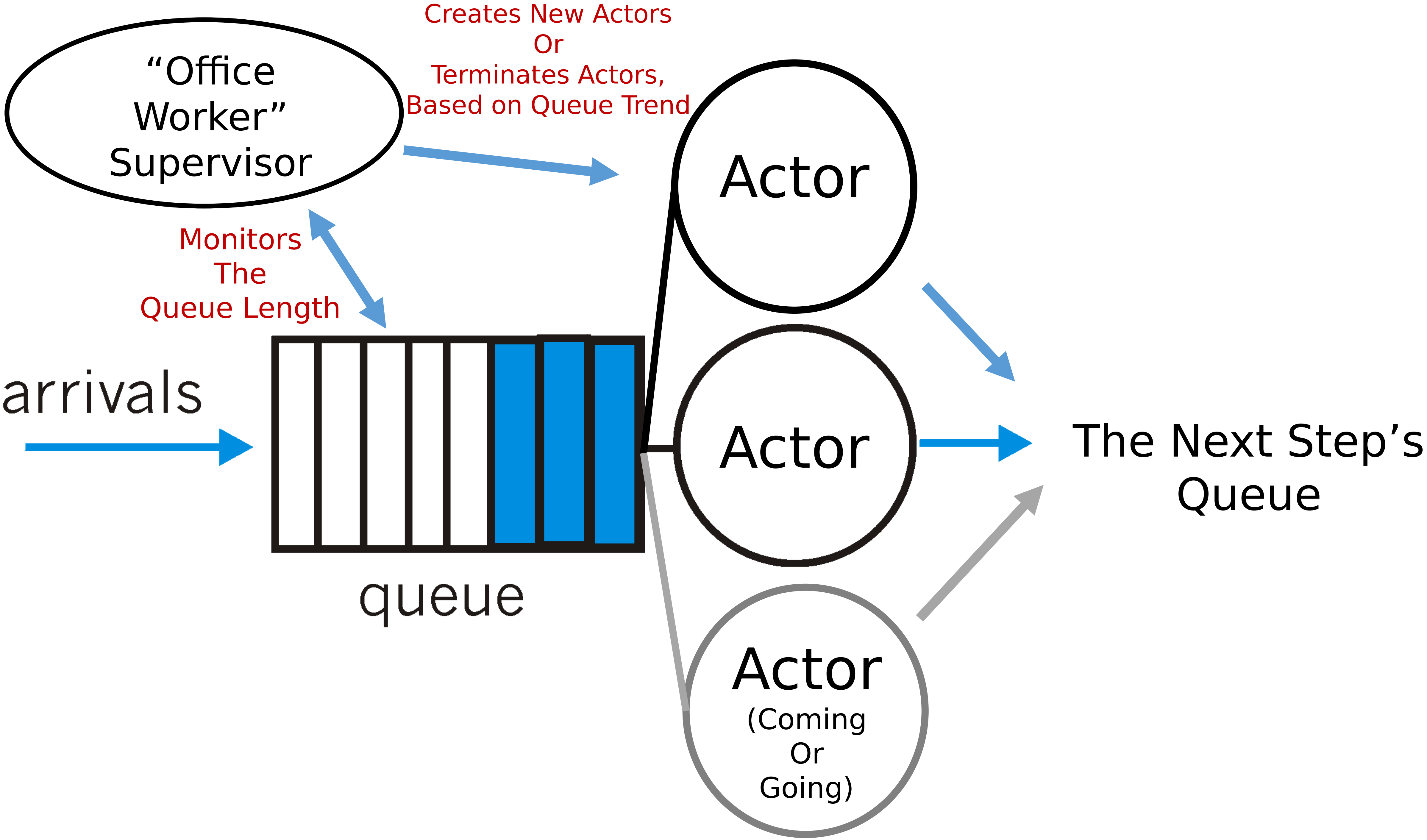
2) P’s and **Queues**

QUEUE

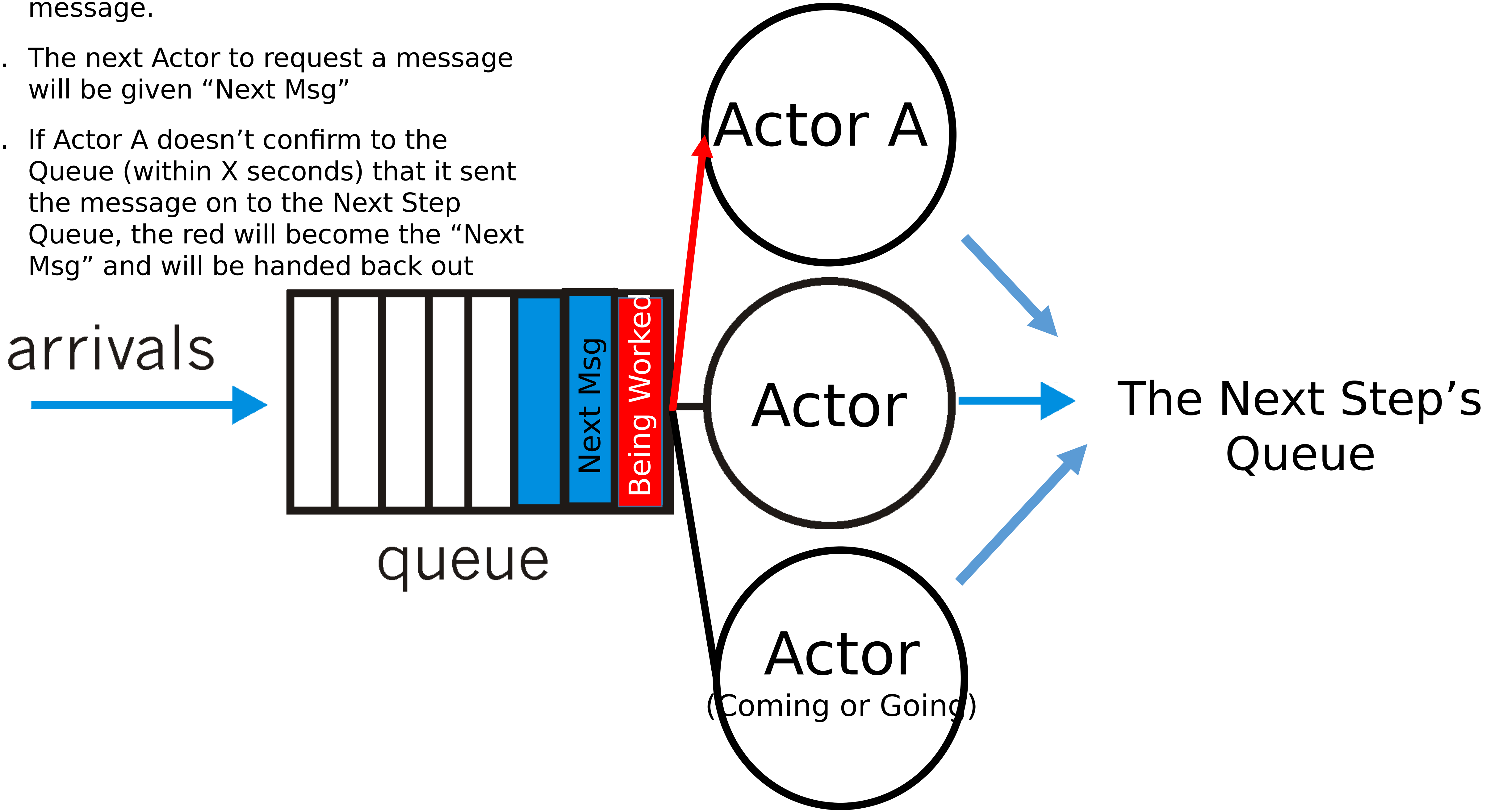



arrivals
Actor
departures
(Infinite) queue


arrivals
queue
Actor
Actor
Actor
(Coming
Or
Going)
The Next Step’s
Queue


"Office Worker" Supervisor
Creates New Actors Or Terminates Actors, Based on Queue Trend
Monitors The Queue Length
arrivals
queue
Actor
Actor
Actor (Coming Or Going)
The Next Step's Queue

1. Actor A is processing the “red” message.
2. The next Actor to request a message will be given “Next Msg”
3. If Actor A doesn’t confirm to the Queue (within X seconds) that it sent the message on to the Next Step Queue, the red will become the “Next Msg” and will be handed back out



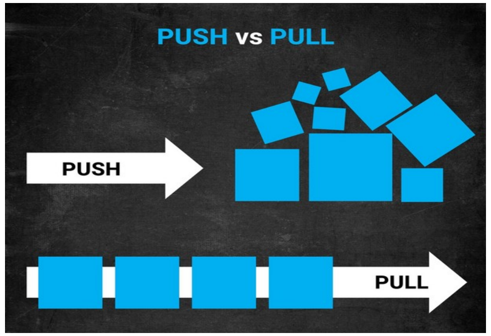
- Use Public Key cryptography to authenticate connection requests from Actors to Queues
- Use Cryptographic Hashes (digital signatures) to both confirm the originator of Messages, and the integrity of the Messages
- Use Encryption to protect the contents of Messages

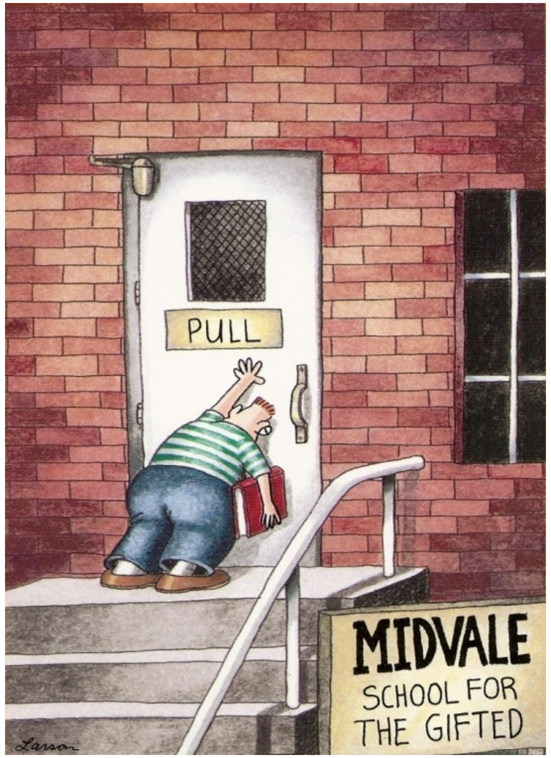
Cryptography is Cheap!!

Use it Everywhere!!

The Four Ingredients in the "Secret Sauce" for Developing Infinitely Scaling High Performance Systems

3)

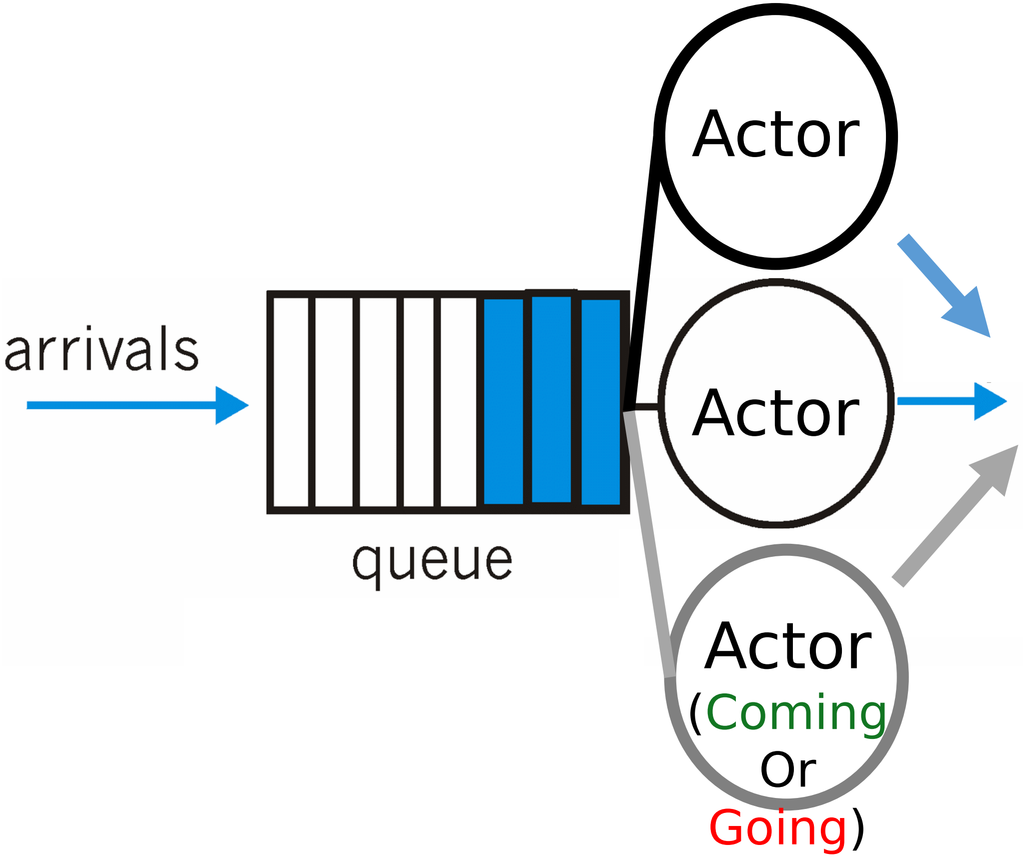




Fudd’s First Law of Opposition: “If you push something hard enough, it WILL fall over.”
- Firesign Theater, 1971

We'll be saying a big hello to all intelligent lifeforms everywhere and to everyone else out there, the secret is to ~~bang the rocks together~~ pull, don’t push, guys.”
- Douglas Adams, Hitchhiker's Guide to the Galaxy

Your Load Balancer
May Push Your Servers
Right Over...



...Instead, Let Your
Applications Pull Their
Workload From a
Common Queue

To Recap...

The keys to building a successful, infinitely scaling application is to:

1. Use the modified Actor Model patterns

    Office Worker
    Supply Clerk
    Supervisor

2. Wire them together with Queues, and

3. Pull, not Push!

Maintaining State

The key to being able to create new instances of Actors to handle the workload is that they must be “stateless”

- Each new message is a new action, without any dependence on previous messages or actions

Don’t

- Don’t store processing state in a database or shared memory
    - A database adds significant complexity to the resiliency and reliability of the application
    - Databases are slow
- Don’t use shared memory
    - This becomes a shared resource that must be protected (with locks, mutexes, etc.)
    - Shared memory doesn’t scale

Maintaining State

Do

- Keep all state information in the message
  When an Actor reads a message, it should have everything it needs to perform it’s task
  This implies that messages are not immutable

Handling External Input

Almost every Real-World Application needs to fetch input from an external source (a source not under its control)

Two choices:

1. Let your Actors block and wait for the fetched information
    - This could easily result in a large number of stalled Actors
    - Incredible resource hog

2. Record the request in a database
    - The database's Supply Clerk Actor is responsible for parsing incoming results and matching them with request records
    - When a result matches up with a request, the Supply Clerk puts a new processing request message in the appropriate queue
    - The Supply Clerk does a periodic scan of outstanding request records. Old records may be an indication of lost requests and may need to be resent
    - Result records without a matching request record are an error and must be flagged to the appropriate Supervisor Actor

- Keep Actors small, with one defined action or decision
- Don’t build a number of small programs and call them Actors
- Encapsulate common resources in Actors – This eliminates locks and race conditions
- Connect Actors together using Queues
- Monitor workloads by monitoring the size of the Queues – Supervisor Actors create new Actors or message surplus Actors to terminate
- Queues only remove acknowledged Messages
- Use “transaction numbers” on truly critical Messages to make certain actions idempotent

- You can “push” into a Queue, but Actors must ALWAYS “pull” from the Queues
- Allow Actors to transparently connect and detach (die) from Queues – no load balancers or registrations
- Keep “state” in the Message between Actors – Each Message is self-sufficient
- Use cryptographic signatures to authenticate Messages and maintain integrity
- Use cryptographic encryption to protect sensitive information in Messages
- Use redundant Actors and Queues for persistence and high availability
- “Map-Reduce”, Paxos, and Raft are your friends

The Four Ingredients in the “Secret Sauce” for Developing Infinitely Scaling High Performance Systems

4) UniK

Unikernels in Light VMs

“Super Containers”

- 1400 Times More Secure Than a Well-Configured Docker Container

- Boots 37 Times Faster Than That Docker Container

- Can Run 10 Times More Microservices (Actors) on the Same Physical Hardware

- Can Be Managed by Kubernetes or Apache Mesos (or an Actor Supervisor)

What Is This Incredible New Technology?

It’s A Unikernel Image Running On A Lightweight Virtual Machine Hypervisor

(If You Are Interested, Come See Me During SELF. If There’s Enough Interest, Perhaps We Can Do a BOF in the Evening)

Carl Hewitt. “Actor Model of Computation for Scalable Robust Information Systems: One IoT is No IoT” 1. Symposium on Logic and Collaboration for Intelligent Applications„ Mar 2017, Stanford, United States. ffhal-01163534v7f, https://hal.science/hal-01163534/document, retrieved 1 June 2023

Zero MQ, An open-source universal messaging library, https://zeromq.org, retrieved 1 June 2023

“SEDA: An Architecture for Well-Conditioned, Scalable Internet Services”, Matt Welsh, David Culler, and Eric Brewer, Computer Science Division University of California, Berkeley {mdw,culler,brewer}@cs.berkeley.edu, http://www.sosp.org/2001/papers/welsh.pdf, retrieved 1 June 2023

** “scaling web applications with message queues” Lenz Gschwendtner, Linux.conf.au 2012 -- Ballarat, Australia https://www.youtube.com/watch?v=aOrGq9yb6og, retrieved 1 June 2023

“My VM is Lighter (and Safer) than your Container”, SOSP ’17, October 28, 2017, Shanghai, China, http://cnp.neclab.eu/projects/lightvm/lightvm.pdf, retrieved 1 June 2023

“Unikernels - Rethinking Cloud Infrastructure”, http://unikernel.org, retrieved 1 June 2023

“Unikernel and Immutable Infrastructure”, https://github.com/cetic/unikernels, retrieved 1 June 2023

** Unik: A Platform for Automating Unikernels Compilation and Deployment, Idit Levine, LISA16 https://www.youtube.com/watch?v=GuRTsCw1Utw, retrieved 1 June 2023

...as well as Wikipedia entries on: Actor Model, Event-Driven Architecture, Message Queue, and Unikernel

** = highly recommended videos

Copies of the slides and the talking points may be downloaded from the Formularity website:

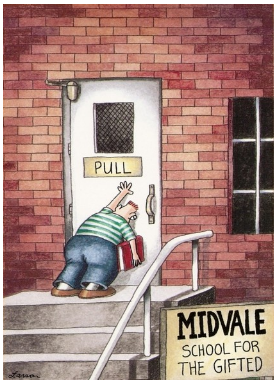
https://formularity.com

“Cartoonist Gary Larson Taught Me How To Build Infinitely Scalable High Performance Systems”

Brad Whitehead

Chief Scientist

Formularity



SouthEast LinuxFest

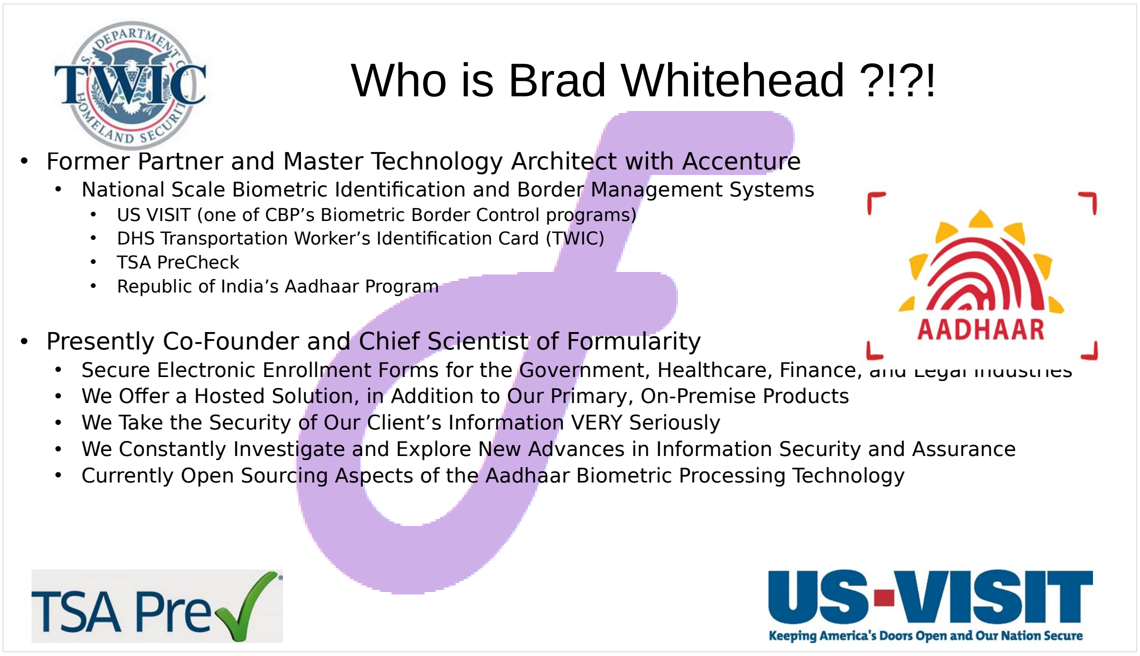
SELF 2023

10 June 2023

Good morning and thank you for taking the time to attend this talk. My name is Brad Whitehead. As the title says, over the next hour, I’m going to tell you how cartoonist Gary Larson’s Farside helped me learn how to build infinitely scaling, high performance systems. But first, I have to apologize. I’m not going to tell you anything new here. You already know what I’m about to tell you. You will leave here thinking “Brad, you didn’t tell me anything I didn’t already know!” However, I ask that you humor me and listen to the presentation anyway.

Before I get started, I want to thank the staff of SouthEast Linux Fest. These volunteers put on a premier event. I’m honored that my presentation was accepted.

So who is Brad Whitehead and why is he qualified to tell me things I already know? I’m the Chief Scientist at Formularity. Formularity is a small company and you may not be familiar with us. We develop high security electronic enrollment forms for things like national identity management programs, financial institutions, and national health care program enrollments. We are dedicated to making sure the sensitive personal information you provide on our forms is secure and protected, even in a cloud environment. In addition to our forms being run by our clients in their data centers, we also offer a hosted solution. Since our target market are national scale enrollments, our hosted solution has a backend infrastructure that can enrollment millions of people per day. Actually, we can scale our backend systems to handle ANY number of enrollments per day. Specifically because we use the architecture I’m going to discuss with you today.



Who is Brad Whitehead ?!?!

- Former Partner and Master Technology Architect with Accenture
  - National Scale Biometric Identification and Border Management Systems
    - US VISIT (one of CBP's Biometric Border Control programs)
    - DHS Transportation Worker's Identification Card (TWIC)
    - TSA PreCheck
    - Republic of India's Aadhaar Program
- Presently Co-Founder and Chief Scientist of Formularity
  - Secure Electronic Enrollment Forms for the Government, Healthcare, Finance, and Legal Industries
  - We Offer a Hosted Solution, in Addition to Our Primary, On-Premise Products
  - We Take the Security of Our Client's Information VERY Seriously
  - We Constantly Investigate and Explore New Advances in Information Security and Assurance
  - Currently Open Sourcing Aspects of the Aadhaar Biometric Processing Technology

Prior to helping to founding Formularity, I was a Partner and Master Technology Architect at Accenture. My specialty, if you will, was national level biometric identification and border management systems. I was Accenture's Chief Architect on US VISIT, the basis of he fingerprint identification system that Customs and Border Protection (CBP) uses when you enter the country, the Transportation Worker's Identification Card (TWIC) system. You need a TWIC card to work at a US Port or to drive a truck with hazardous chemicals, like gasoline. And TSA Pre-Check. I'm sure everybody here is familiar with Pre-Check. These are all fingerprint identification systems. They are also "guns and badges" systems, where the Government tells you what you and can't do, based on your fingerprints. However, my biggest source of pride is having been Accenture's Chief Architect on the largest human rights and social enablement program in the world. This is the Republic of India's Aadhaar system. I apologize for my poor pronunciation of Aadhaar. I mention Aadhaar and people say that they have never heard of it. Then they see the logo and say "Oh, you mean Aadhaar!". My poor Ango-Saxon ears can't hear the difference.

UIDAI
Unique Identification Authority of India
Planning Commission, Government of India
AADHAAR
Aadhaar is now the identity of 100 crore residents of India
Shri Narendra Modi
Hon'ble Prime Minister
Aadhaar given to 100 crore residents
93% of adults have Aadhaar
25.48 crore bank accounts linked to Aadhaar
12.28 crore LPG consumers linked with Aadhaar
11.39 crore ration cards linked to Aadhaar
5.9 crore Aadhaar seeded in MGNREGS database
Aadhaar powering Digital India- Digi Locker, eSign, etc.
Shri Ravi Shankar Prasad
Union Minister for Communications & IT congratulates and shares the proud moment with the nation
Image Credit: uidai.gov.in
Unique Identification Authority of India
Planning Commission, Government of India
ALWAYS CARRY YOUR AADHAAR WITH YOU
and avail benefits of various services
4444 5555 6666
Cut this portion of Aadhaar letter and carry it with you.
(You may get it laminated too)
Aadhaar : your proof of identity and proof of address for
...and many more services
AADHAAR
www.uidai.gov.in

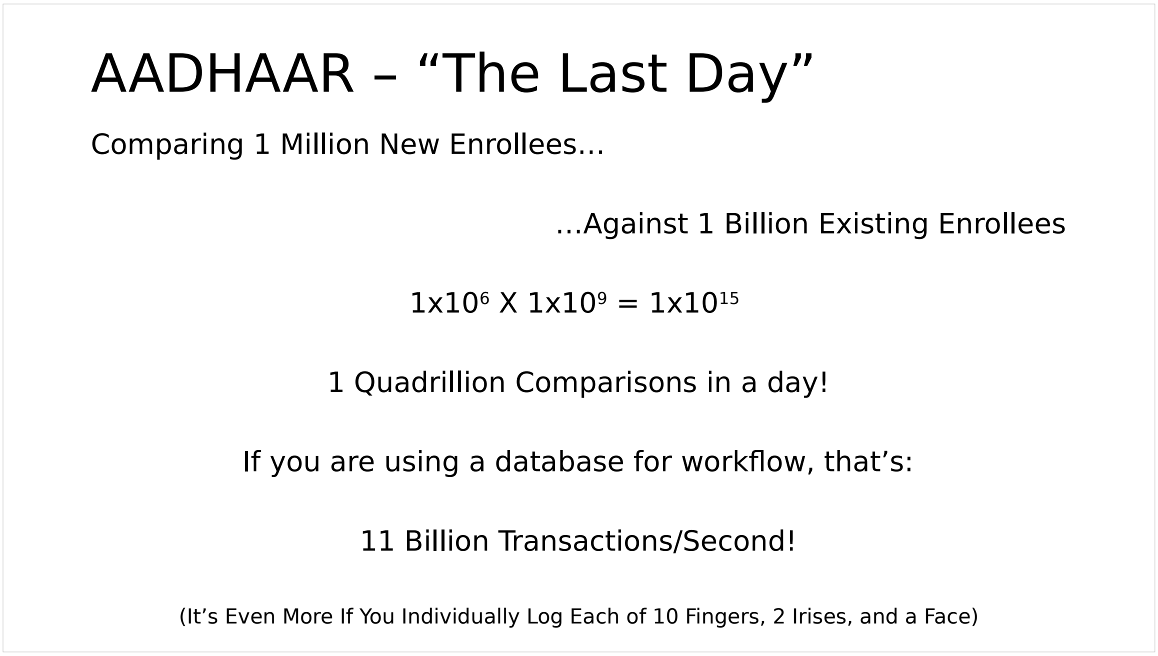
So, is anybody familiar with Aadhaar? India has the second largest population in the world – approximately 1.1 billion people. It is also one of the wealthest countries. Unfortunately, that wealth is very unevenly distributed. 80% of the population falls below the poverty line. Fortunately, India is very socially responsible and has large social programs to help the poorer citizens with food rations, cooking gas, guaranteed employment, etc. However, wherever you have lots of cash being distributed, you have lots of corruption and graft.

Many of the people for whom these programs are intended have no “official” identity. No drivers license, no passport, often no fixed address. These people are then easily cheated out of their Government rights. The Unique Identification Authority of India (UIDAI) was founded under the Gandhi/Singh Congress Party administration and it continues under the Modi BJP administration. UIDAI’s Aadhaar program voluntarily enrolled over one billion Indians into a biometrically-based identification system. No longer can a corrupt supervisor fail to pay his or her workers. These workers can now open bank accounts and have their wages directly deposited, with an audit trail.

Likewise, a rice merchant can’t give rice to his or her friends and then claim it went to people with ration cards.

As part of Aadhaar, we registered all 10 fingers and both irises of over one billion Indian residents. Actually, collecting and enrolling the biometrics was the easy part. After we collected a set of biometrics, we had to compare the new set with every other set that had already been enrolled, to keep people from accidentally or intentionally creating two or more identities

Accenture was one of the three original Biometric Service Providers for Aadhaar. We had to process 1 million enrollments per day. Day



AADHAAR – “The Last Day”

Comparing 1 Million New Enrollees…

…Against 1 Billion Existing Enrollees

$1x10^6$ X $1x10^9$ = $1x10^{15}$

1 Quadrillion Comparisons in a day!

If you are using a database for workflow, that’s:

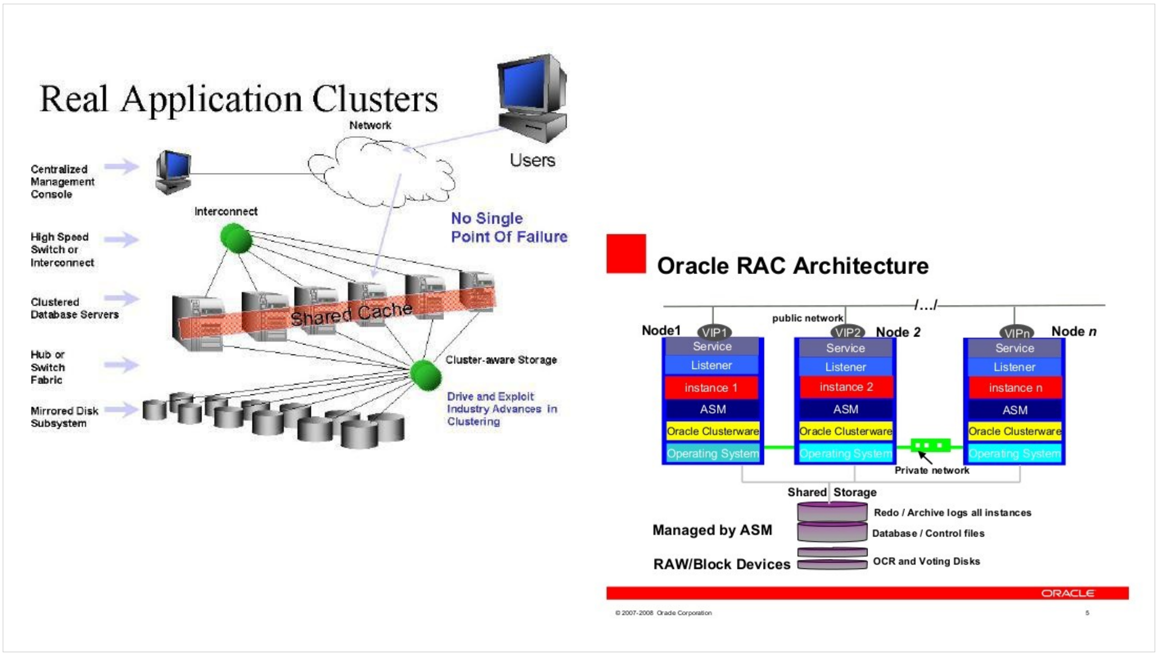
11 Billion Transactions/Second!

(It’s Even More If You Individually Log Each of 10 Fingers, 2 Irises, and a Face)

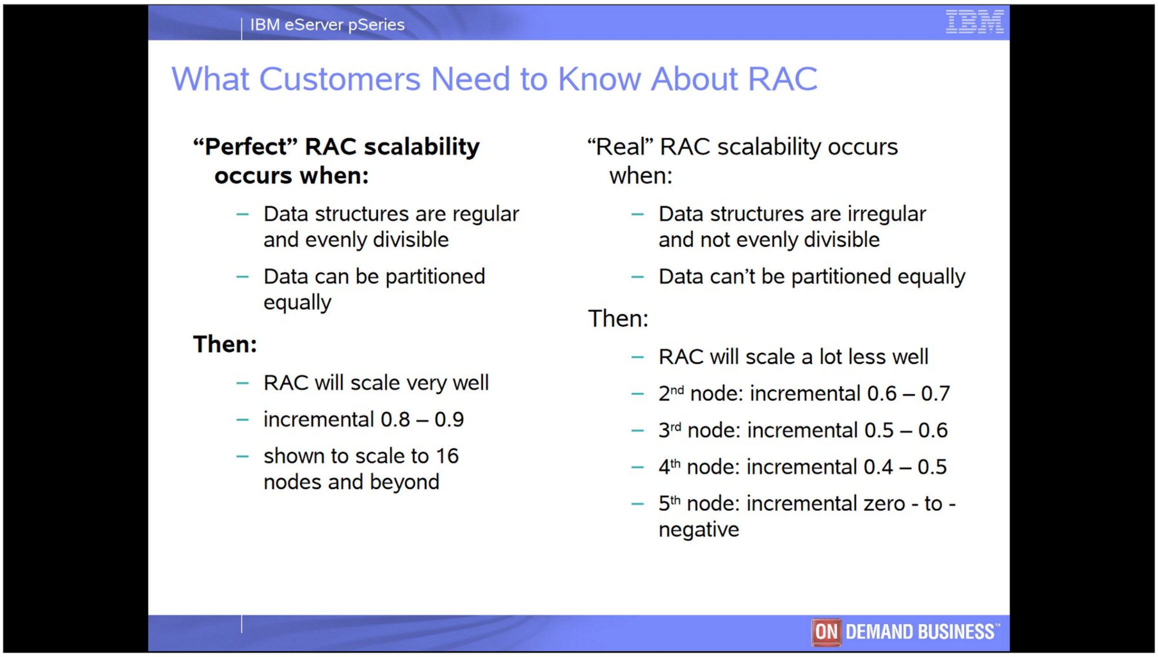
Now, there is no indexing or hashing system for fingerprints. The system must compare each fingerprint or iris print against every existing print. What a database person would call a full table scan. Now, consider the “last day” of Aadhaar – we have to compare our daily 1 million enrollees against the 1 billion existing enrollees! 1 Quadrillion comparisons or 11 billion transactions per second!

I took over as Acccenture’s Chief Architect on the program when we ran into scaling issues. Initially, we started by using the industry leading biometric middleware. This product uses an Oracle database as its workflow engine.

When we started processing over 500 thousand biometric enrollments per day, we hit the transaction limits of the Oracle system. And I don’t mean the limits for our particular hardware infrastructure, I mean the absolute limits of the Oracle database. Oracle scaled, at the time, using an architecture they call Real Application Clusters.



Basically, you have any number of Oracle databases running on separate servers, working together. Theoretically, you can have any number of servers in the cluster and it can scale to any workload. However, with real-world data loads, this infinite scaling doesn’t happen. It’s difficult to get good benchmarks and case studies on Oracle products because Oracle clients are prohibited by their license from publishing any type of benchmarking information,



IBM eServer pSeries

What Customers Need to Know About RAC

"Perfect" RAC scalability occurs when:

- Data structures are regular and evenly divisible
- Data can be partitioned equally

Then:

- RAC will scale very well
- incremental 0.8 – 0.9
- shown to scale to 16 nodes and beyond

"Real" RAC scalability occurs when:

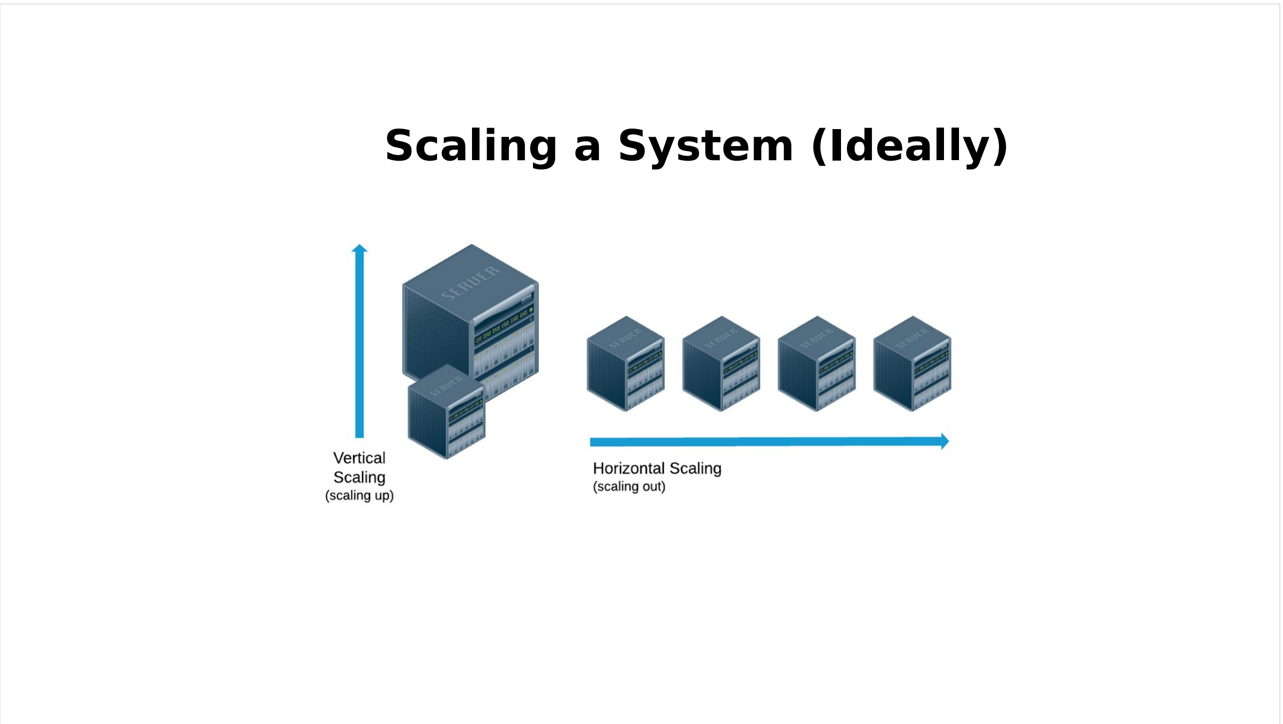
- Data structures are irregular and not evenly divisible
- Data can't be partitioned equally

Then:

- RAC will scale a lot less well
- 2nd node: incremental 0.6 – 0.7
- 3rd node: incremental 0.5 – 0.6
- 4th node: incremental 0.4 – 0.5
- 5th node: incremental zero - to - negative

ON DEMAND BUSINESS™

…but one Oracle partner has published data that shows that for the average real-world workload, most Oracle Real Application Cluster systems stop scaling after 5 nodes in the cluster. This was certainly my experience in India. We had 4 nodes in the cluster and we couldn't push more transactions through the system by adding more nodes or by tuning the nodes we had. And when I say "we", I mean Oracle themselves. We had Oracle's own developers from both Redwood Shores and Bangalore helping us. My first job as Chief Architect was to replace the Oracle database-driven workflow system with a new workflow system using the methodologies I'm talking to you about today. This workflow system easily handles 1 million enrollments per day and since we scale linearly, we can dial in how many enrollments by just increasing the number of servers.



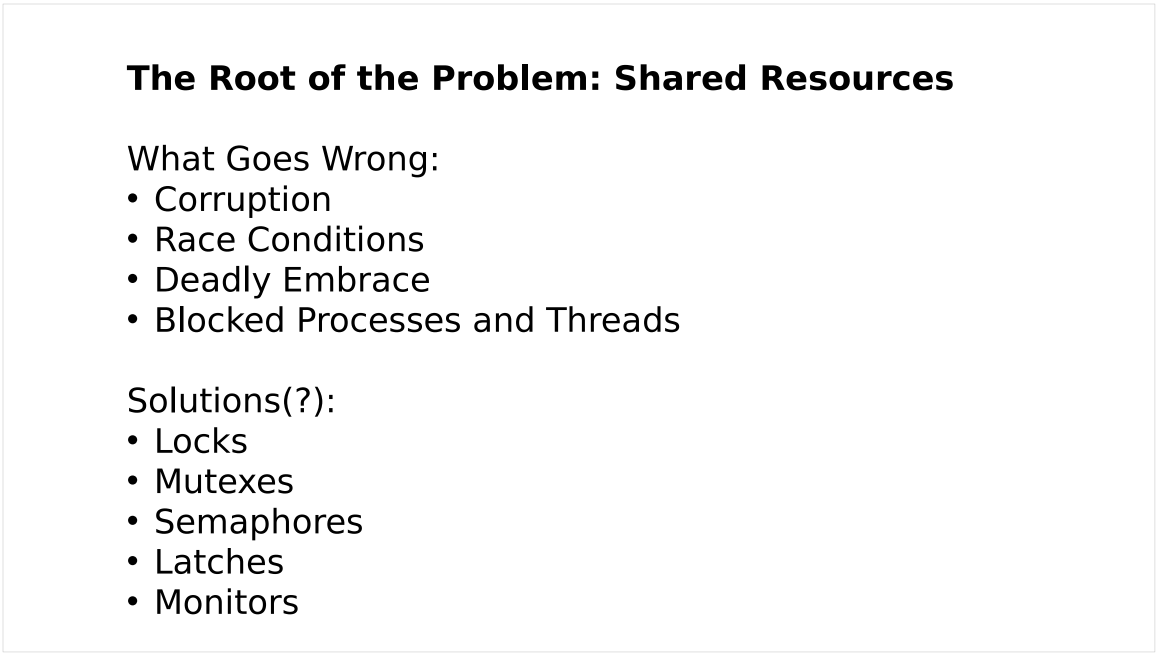
Scaling a System (Ideally)

Vertical
Scaling
(scaling up)

Horizontal Scaling
(scaling out)

Here’s the traditional “scaling slide”. We start with vertical scaling, using one server and beefing it up as much as possible. Soon, we exhaust the scaling capabilities of a single computer. We are clocking the processors as fast as we can, and we are using all its cores and hyper-threads. So we go to horizontal scaling. We add multiple computers clustered together to increase the parallelism and apparent speed and scale of the application.

At least we are finally, we are on the right track. Independent computers offer more resiliency and availability.

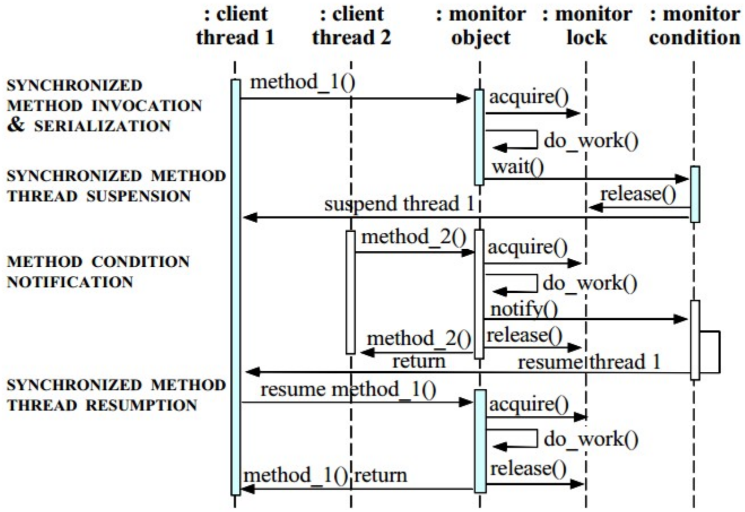


The Root of the Problem: Shared Resources

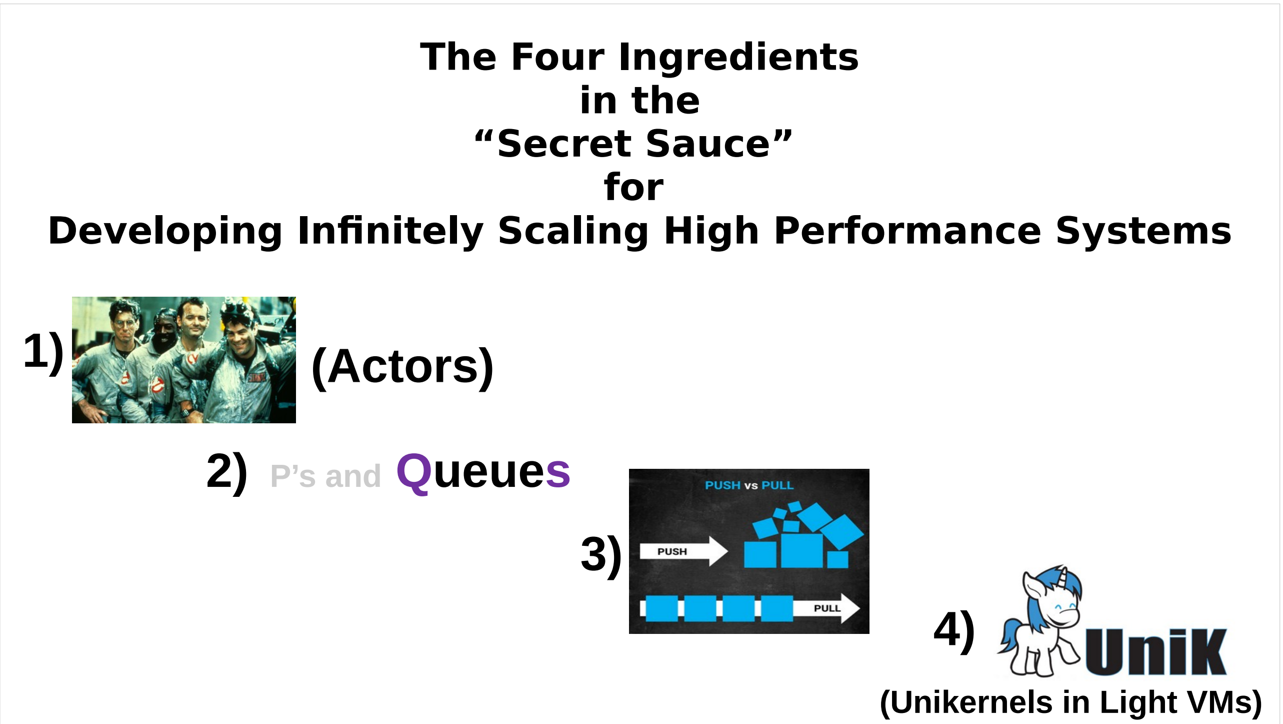
What Goes Wrong:
- Corruption
- Race Conditions
- Deadly Embrace
- Blocked Processes and Threads

Solutions(?):
- Locks
- Mutexes
- Semaphores
- Latches
- Monitors

So, what went wrong with Real Application Clusters and its horizontal scaling? Even though each server is separate, the overall database has shared resources. If we don’t protect the shared resources, we risk corruption and deadlock. So we have developed “simple” protection mechanisms, like locks and mutexes.



This diagram shows a “simple” (“air quotes”) thread synchronization mechanism. I see states like “suspend” and “wait”, not what I want in a scaling system!



The Four Ingredients in the “Secret Sauce” for Developing Infinitely Scaling High Performance Systems

1) (Actors)

2) P’s and Queues

3)

4) UniK

(Unikernels in Light VMs)

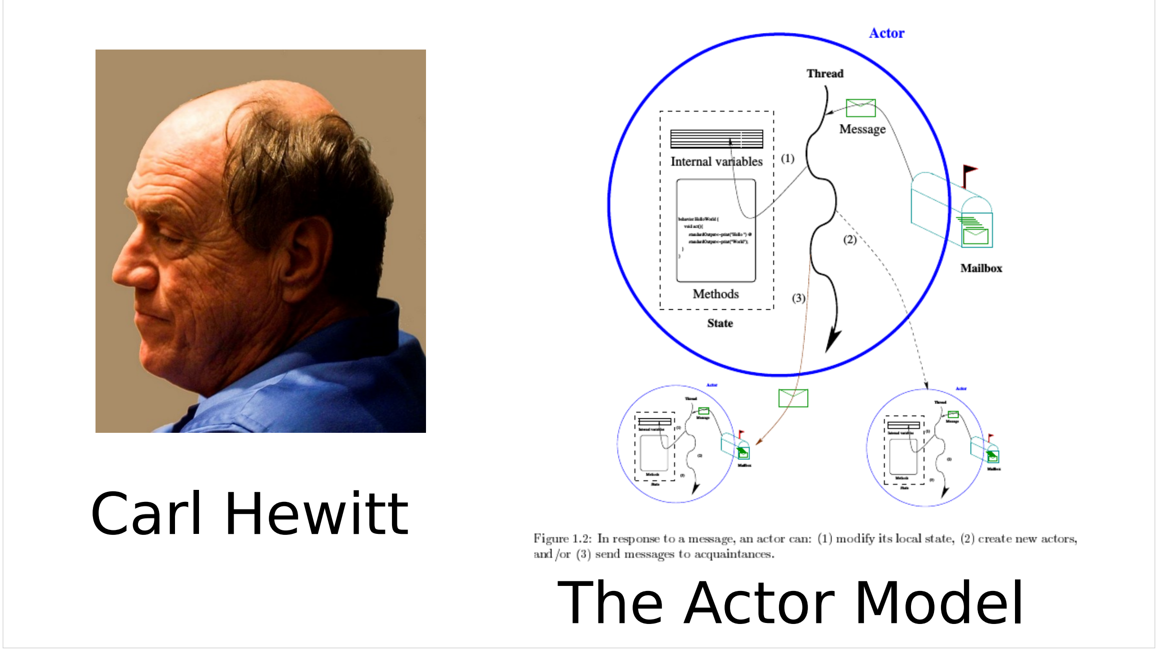
There are four “ingredients” in the “secret sauce” for developing infinitely scaling, high performance systems like both the Aadhaar and Formularity enrollment systems.

In keeping with our “I Ain’t Afraid of No Bug” theme – Actors, Queues, Pull, don’t Push, and unikernels running on lightweight virtual machines. BTW, I’m using Unique’s logo since there’s no industry logo for unikernels, Solo-io makes one of the best unikernel construction toolkits...and their unicorn is cute :-)

The Four Ingredients in the "Secret Sauce" for Developing Infinitely Scaling High Performance Systems

1) (Actors)

Let's start with Actors – the information processing paradigm, not Ramis, Murray, and Aykroyd

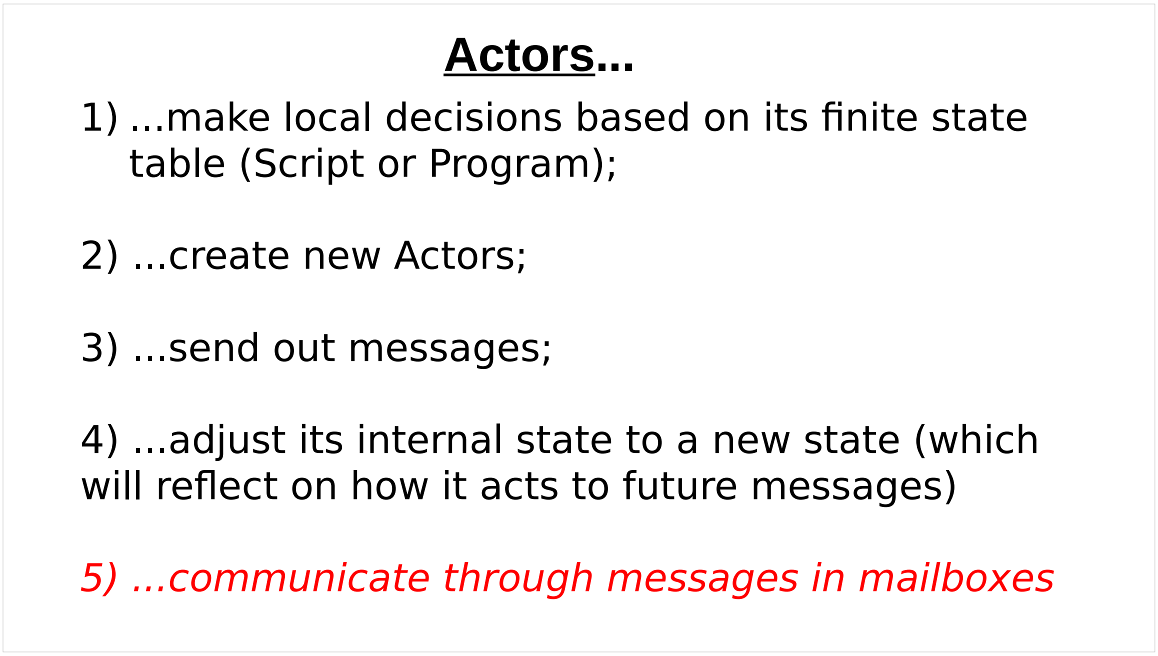


Carl Hewitt

Figure 1.2: In response to a message, an actor can: (1) modify its local state, (2) create new actors, and/or (3) send messages to acquaintances.

The Actor Model

In 1973, Carl Hewitt published the Actor Model of concurrency. Basically, an Actor is an automation that receives external input through messages. Based on a received message, an Actor automation can do one of five things, it can::



Actors...

1) ...make local decisions based on its finite state table (Script or Program);

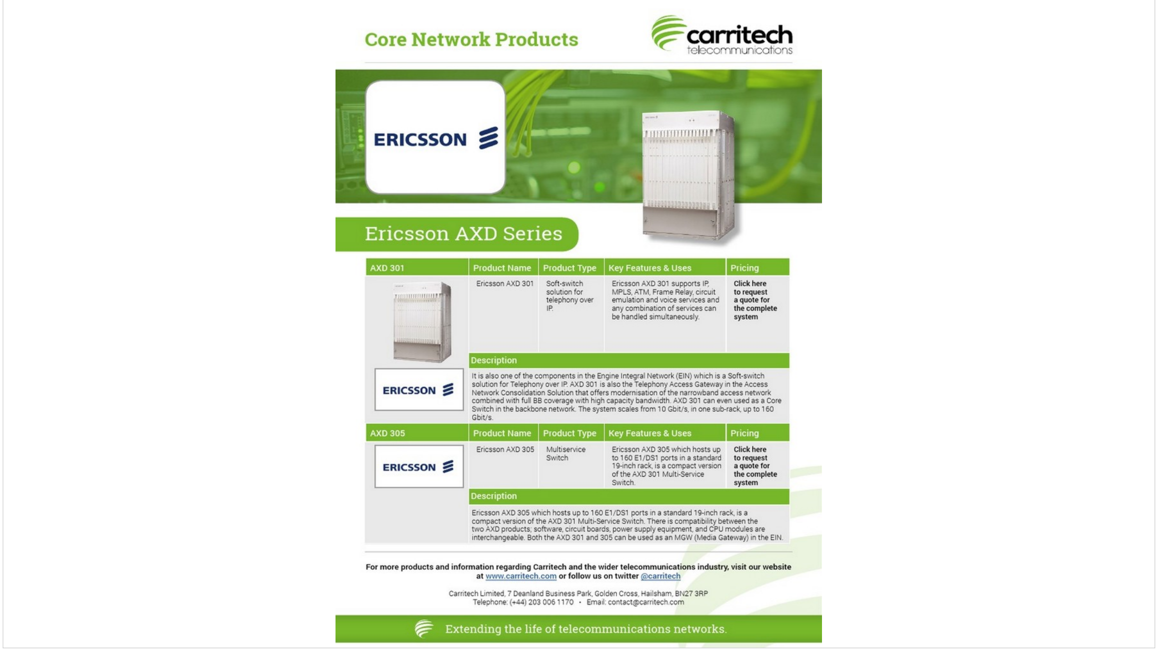
2) ...create new Actors;

3) ...send out messages;

4) ...adjust its internal state to a new state (which will reflect on how it acts to future messages)

5) ...communicate through messages in mailboxes

1) Make a decision based on its programming or script. 2) It can replicate and create a new Actor. 3) It can send out a message to another Actor, 4) it can change its internal settings, and 5) it communicates through messages in a mailbox. So, I’ve noted the last item in red, because is where I disagree with Hewitt and where we’ve changed his original Actor model. I’ll explain in a slide or so. While Hewitt’s model was mathematical and theoretical, practical instances of Actors can and have been written. For example, the programming language Erlang is Actor-based.



Core Network Products

carritech
telecommunications

Ericsson AXD Series

AXD 301	Product Name	Product Type	Key Features & Uses	Pricing
ERICSSON	Ericsson AXD 301	Soft-switch solution for telephony over IP.	Ericsson AXD 301 supports IP, MPLS, ATM, Frame Relay, circuit emulation and voice services and any combination of services can be handled simultaneously.	**Click here to request a quote for the complete system**

Description

It is also one of the components in the Engine Integral Network (EIN) which is a Soft-switch solution for Telephony over IP. AXD 301 is also the Telephony Access Gateway in the Access Network Consolidation Solution that offers modernisation of the narrowband access network combined with full BB coverage with high capacity bandwidth. AXD 301 can even used as a Core Switch in the backbone network. The system scales from 10 Gbit/s, in one sub-rack, up to 160 Gbit/s.

AXD 305	Product Name	Product Type	Key Features & Uses	Pricing
ERICSSON	Ericsson AXD 305	Multiservice Switch	Ericsson AXD 305 which hosts up to 160 E1/DS1 ports in a standard 19-inch rack, is a compact version of the AXD 301 Multi-Service Switch.	**Click here to request a quote for the complete system**

Description

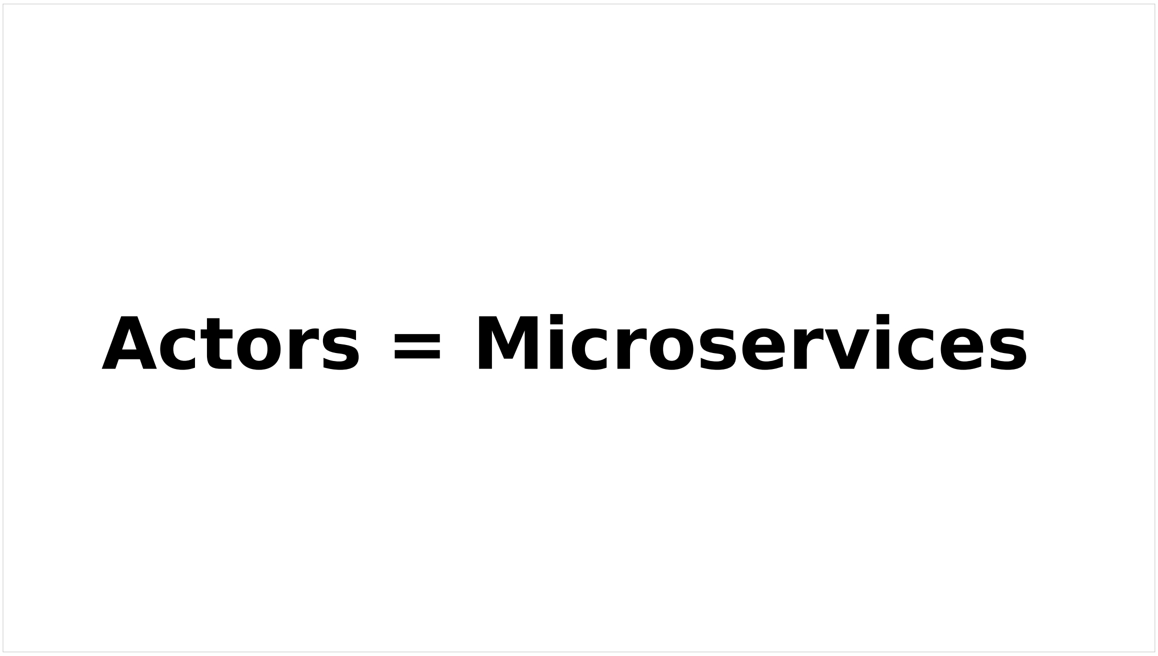
Ericsson AXD 305 which hosts up to 160 E1/DS1 ports in a standard 19-inch rack, is a compact version of the AXD 301 Multi-Service Switch. There is compatibility between the two AXD products; software, circuit boards, power supply equipment, and CPU modules are interchangeable. Both the AXD 301 and 305 can be used as an MGW (Media Gateway) in the EIN.

For more products and information regarding Carritech and the wider telecommunications industry, visit our website at www.carritech.com or follow us on twitter @carritech

Carritech Limited, 7 Deanland Business Park, Golden Cross, Hailsham, BN27 3RP
Telephone: (+44) 203 006 1170 • Email: contact@carritech.com

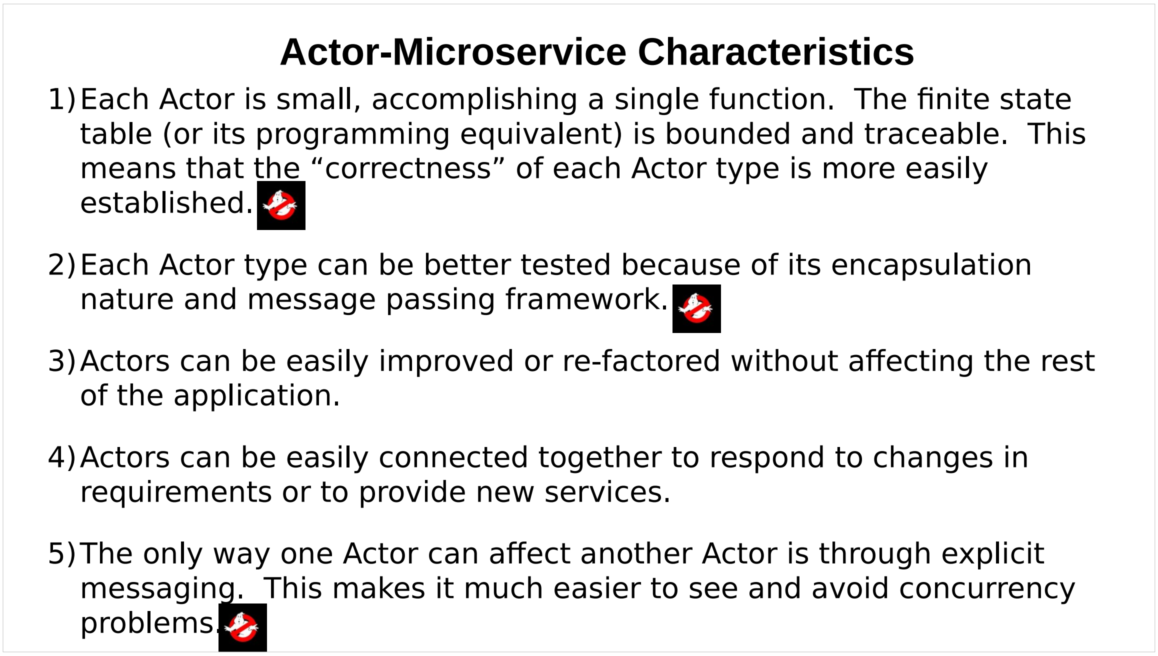
Extending the life of telecommunications networks.

Designed to handle the thousands of simultaneous conversations in a telephone switch, Erlang and the Actor Model are directly responsible for the Erisson AXD310 telephone having a stated high availability of 9 “9s”. That’s a down time of 31 milliseconds per year. That’s better than anything I’ve ever achieved in a Tier IV Data Center!



Actors = Microservices

So, Actors are a perfect working definition of a microservice. What’s the definition of a microservice? Who knows! There is no formal definition and everybody has their own opinion. I personally like the Actor model as a definition of a microservice.



Actor-Microservice Characteristics

1) Each Actor is small, accomplishing a single function. The finite state table (or its programming equivalent) is bounded and traceable. This means that the "correctness" of each Actor type is more easily established.

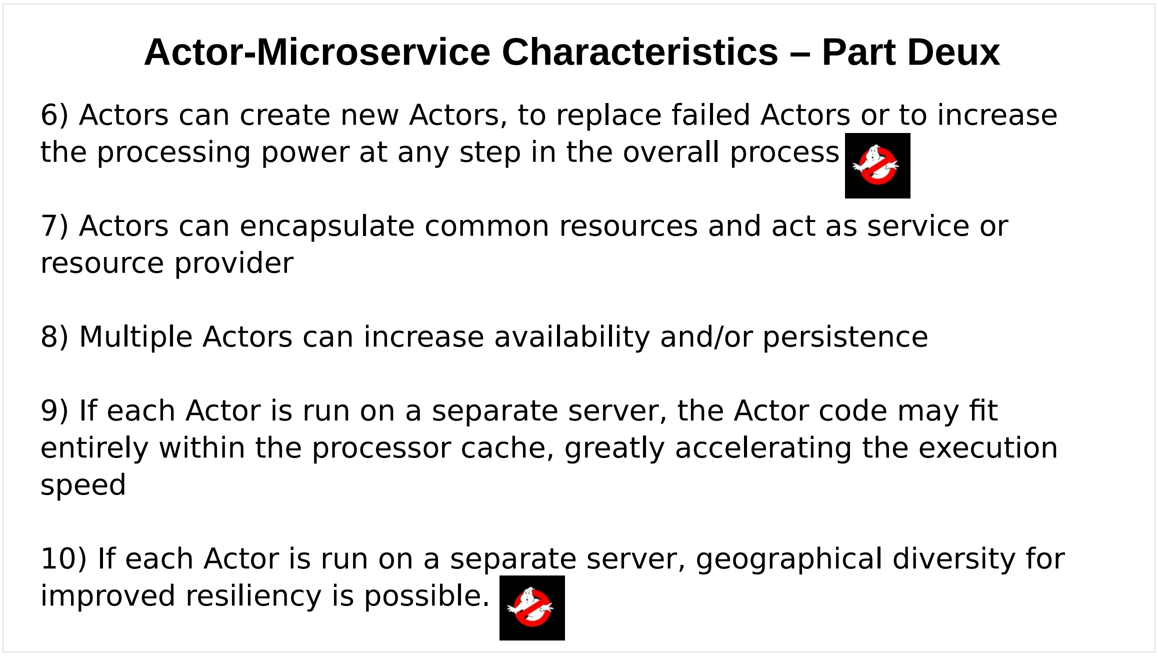
2) Each Actor type can be better tested because of its encapsulation nature and message passing framework.

3) Actors can be easily improved or re-factored without affecting the rest of the application.

4) Actors can be easily connected together to respond to changes in requirements or to provide new services.

5) The only way one Actor can affect another Actor is through explicit messaging. This makes it much easier to see and avoid concurrency problems.

Again, in homage to our theme this year, I've denoted those characteristics that help in producing error-free code



Actor-Microservice Characteristics – Part Deux

6) Actors can create new Actors, to replace failed Actors or to increase the processing power at any step in the overall process

7) Actors can encapsulate common resources and act as service or resource provider

8) Multiple Actors can increase availability and/or persistence

9) If each Actor is run on a separate server, the Actor code may fit entirely within the processor cache, greatly accelerating the execution speed

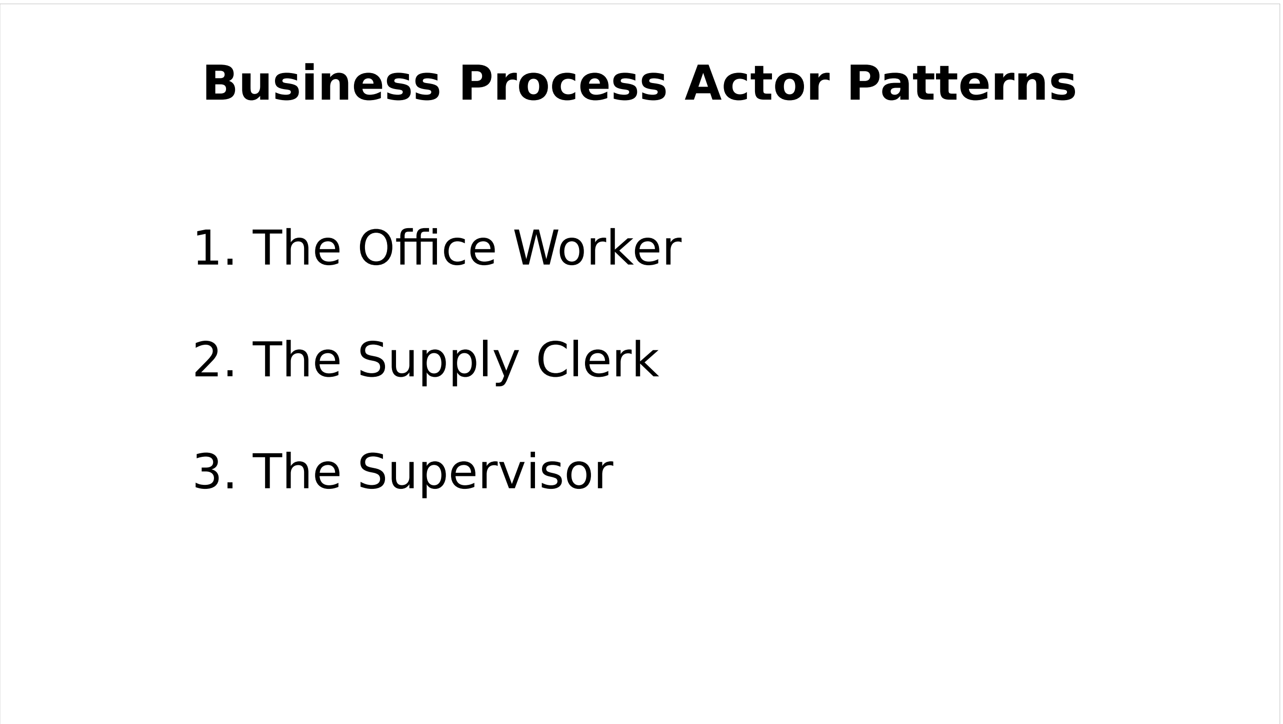
10) If each Actor is run on a separate server, geographical diversity for improved resiliency is possible.

Actors as a definition for microservice has so many advantages, I had to put them on two slides

Actors – “Good Stuff!

“Actors – Good Stuff!!! (double thumbs up)

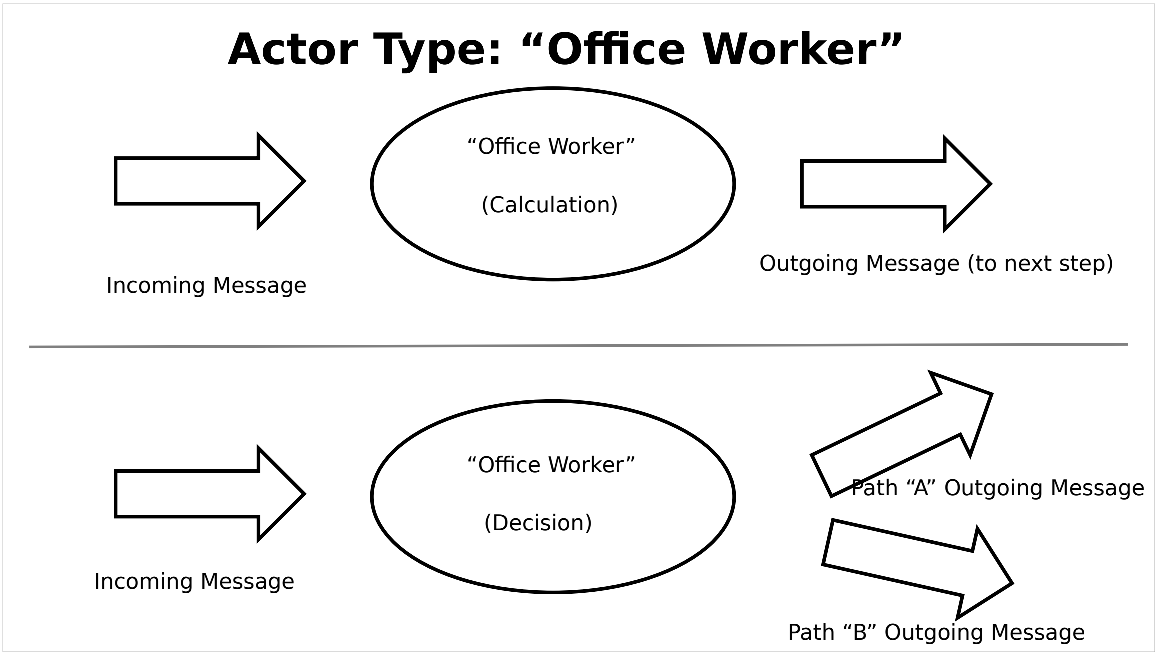
It has been my experience that human organization quite often provides the best way of doing something. So my first step is always to identify how a person would do a given task and then see how it can be applied to an automated system. This isn’t foolproof, but it’s a good starting point. So, if we consider our Actors to be actual people, what services are necessary and how can we implement them?



Business Process Actor Patterns

1. The Office Worker
2. The Supply Clerk
3. The Supervisor

We’ve identified three business process Actor patterns: the Office Worker, the Supply Clerk, the Supervisor



Actor Type: “Office Worker”

Incoming Message

“Office Worker”

(Calculation)

Outgoing Message (to next step)

Incoming Message

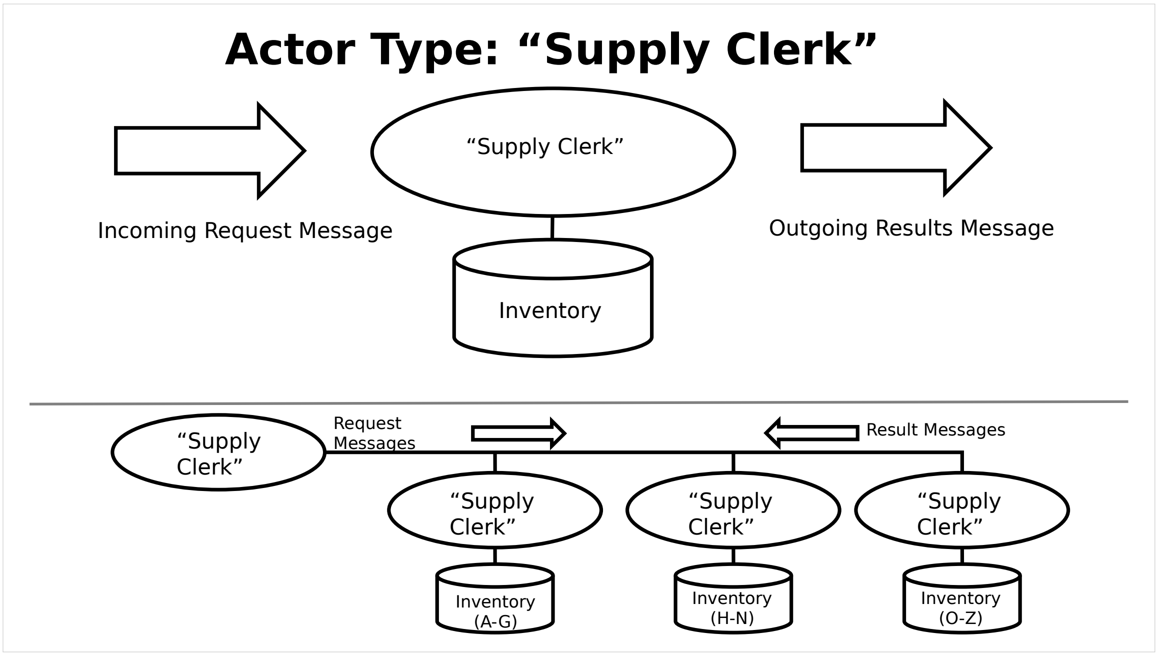
“Office Worker”

(Decision)

Path “A” Outgoing Message

Path “B” Outgoing Message

So, in a corporate organization, who does the work? An office worker. He or she sits there, picks a file out of the Inbox, does something to it, and puts the results into one or more Outboxes. Our Office Worker Actor is one of three patterns that form the bricks of our business process. It takes in a message, performs a calculation or transformation on the message, using its internal programming and state, and outputs the results. It might be making a decision and therefore it will send the results out to one or more other Actors



Actor Type: “Supply Clerk”

Incoming Request Message

“Supply Clerk”

Inventory

Outgoing Results Message

“Supply Clerk”

Request Messages

Result Messages

“Supply Clerk”

Inventory (A-G)

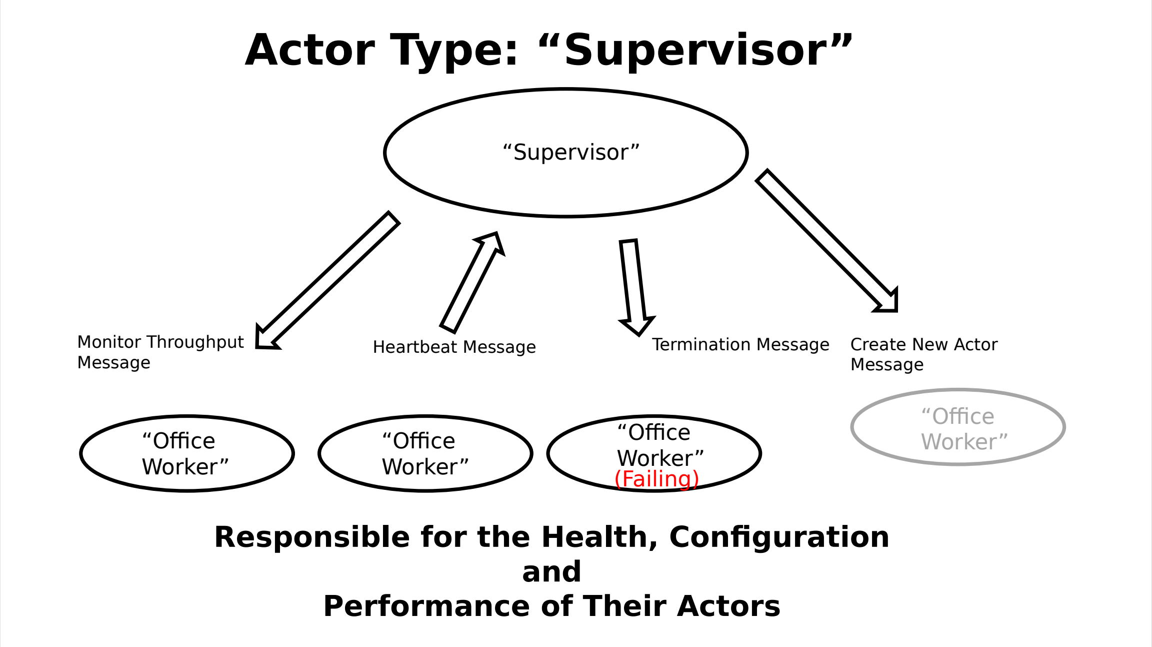
“Supply Clerk”

Inventory (H-N)

“Supply Clerk”

Inventory (O-Z)

Our next Actor pattern is the Supply Clerk. The Supply Clerk-type Actor encapsulates common resources. Need to update inventory? Don’t allow an Office Worker Actor to change the inventory. If you do, two or more Actors will make conflicting changes. Make the inventory the responsibility of a single Supply Clerk Actor instance. If the inventory needs to be updated, send the update request to the Supply Clerk Actor. Problem solved. So what if you have too many inventory changes per second for one Supply Clerk? We have a couple of different choices: We can use an Hadoop-like “scatter-gather” or the “map-reduce” configuration with multiple Supply Clerk Actors. Or, much like the database itself, we can shard the Supply Clerks. A Office Worker with a Distributed Hash Table (DHT) could provide routing information for requests to multiple Supply Clerks. We can definitely use multiple Supply Clerks for redundancy, persistence, and consensus. As the architect, you have to make the best decision based on the requirements. The flexibility of the Actor Model doesn’t lock you out of any options.



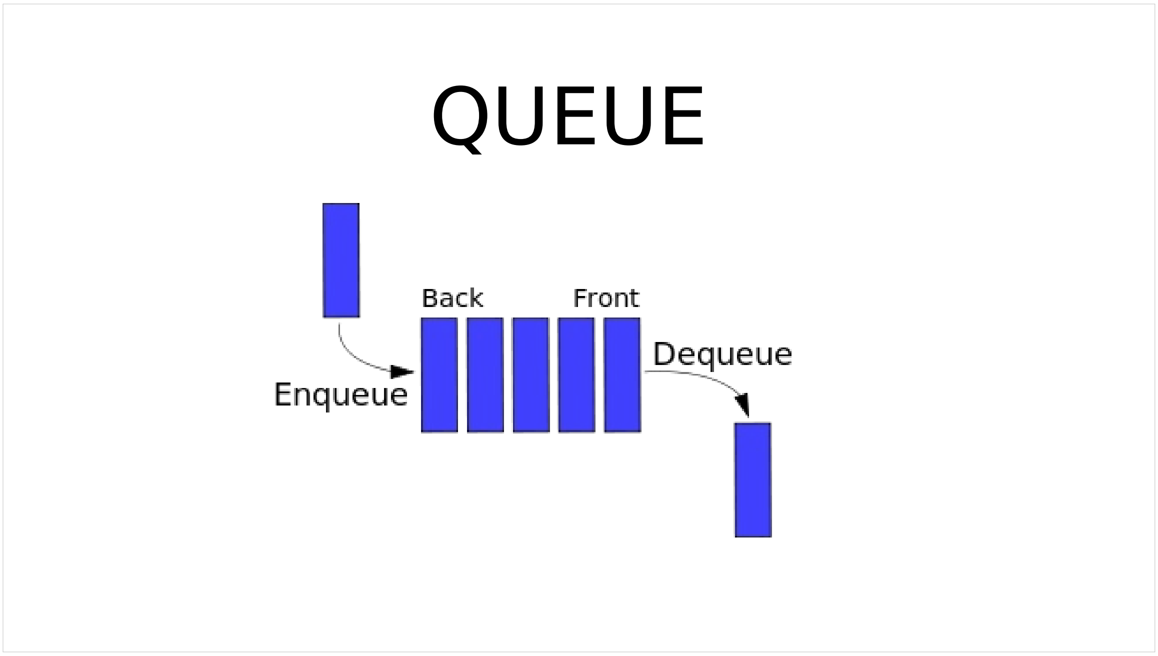
Actor Type: “Supervisor”

Responsible for the Health, Configuration and Performance of Their Actors

Our third pattern is the Supervisor. The Supervisor is responsible for the overall application running smoothly and meeting its processing time requirements. The Supervisor monitors the health of all the Office Workers and Supply Clerks under its supervision. It can do this by observing the throughputs of the subordinate Actors, or it can actively ping subordinate Actors, or by receiving “heartbeat” messages from subordinates. Regardless of how it monitors the other Actors, it is responsible for messaging failing Actors to gracefully terminate themselves (after they complete their current process cycle), or killing the operating system process of the failed Actor. The Supervisor Actor monitors the performance of the processing cluster for which it’s responsible and for 2) starting new Actors to handle increased workloads, or 2) messaging surplus Actors to terminate when the workload decreases. How it does this workload monitoring is critical and is dependent on the second ingredient in our Secret Sauce. I’ll talk about that next. However, just to finish up on the Supervisor Actor, Supervisor Actors definitely are hierarchical in nature, with Supervisors supervising subordinate Supervisors.

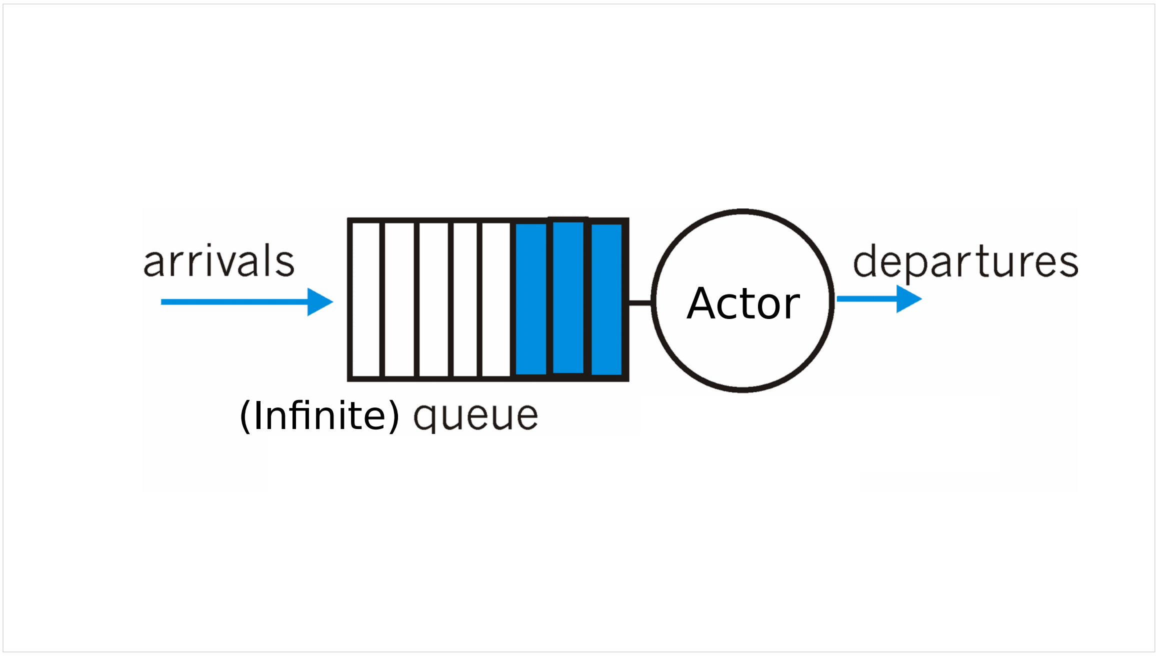
The Four Ingredients
in the
“Secret Sauce”
for
Developing Infinitely Scaling High Performance Systems

2) P’s and **Queues**

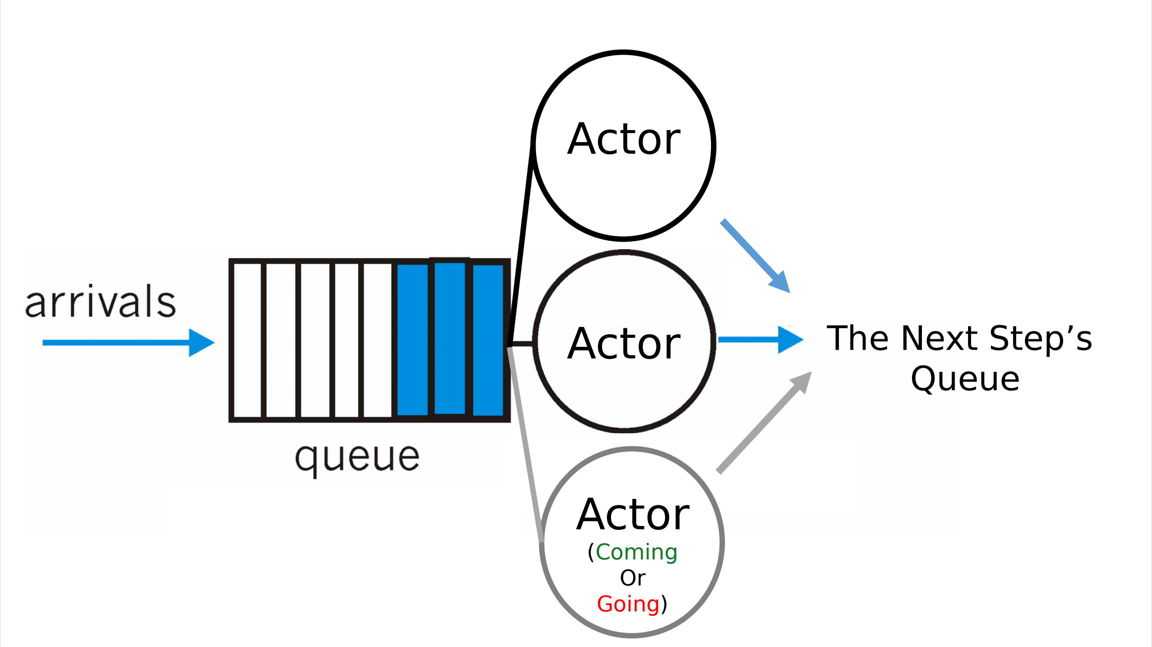


QUEUE

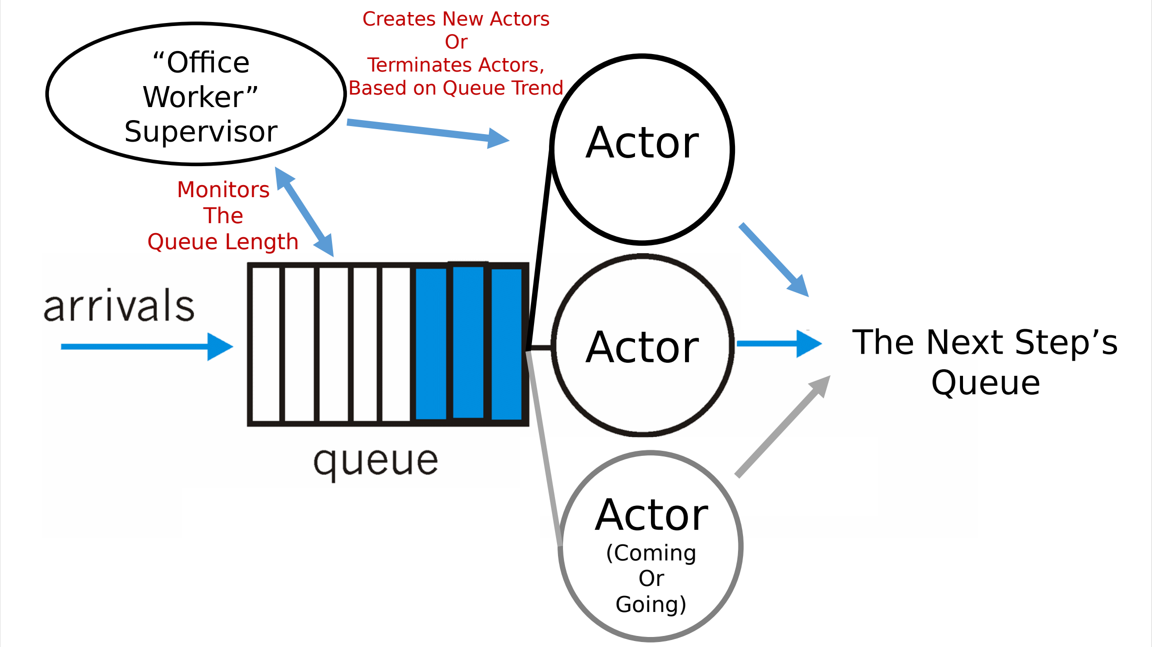
Queues, or ordered lists are a basic storage structure in Computer Science. In our case, the Queue is a FIFO stack – “First in is First Out”, and what we are storing are messages between Actors.



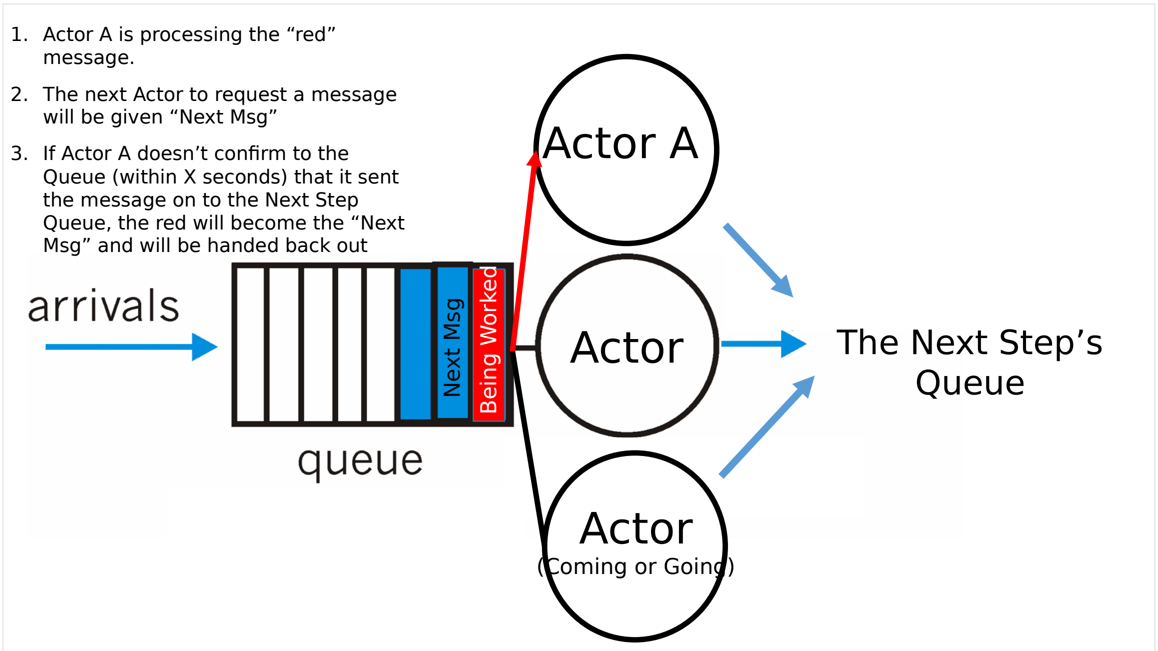
In Hewitt’s original Actor model, the Actors used mailboxes, sending messages to other Actor’s mailboxes and receiving messages through their own mailbox. A mailbox is just a type of Queue. So far, we are in-sync with the original Actor model as Hewitt envisioned it.



In fact, this is exactly how you efficiently manage the number and types of Actors in your application. Watch the size of a Queue. If messages are backing up and the Queue is getting larger, you need to start up more Actors and attach them to that Queue. By the same token, if the Queue is shrinking or empty, you have too many Actors attached to it. Have the monitoring Supervisor terminate several of the surplus Actors.

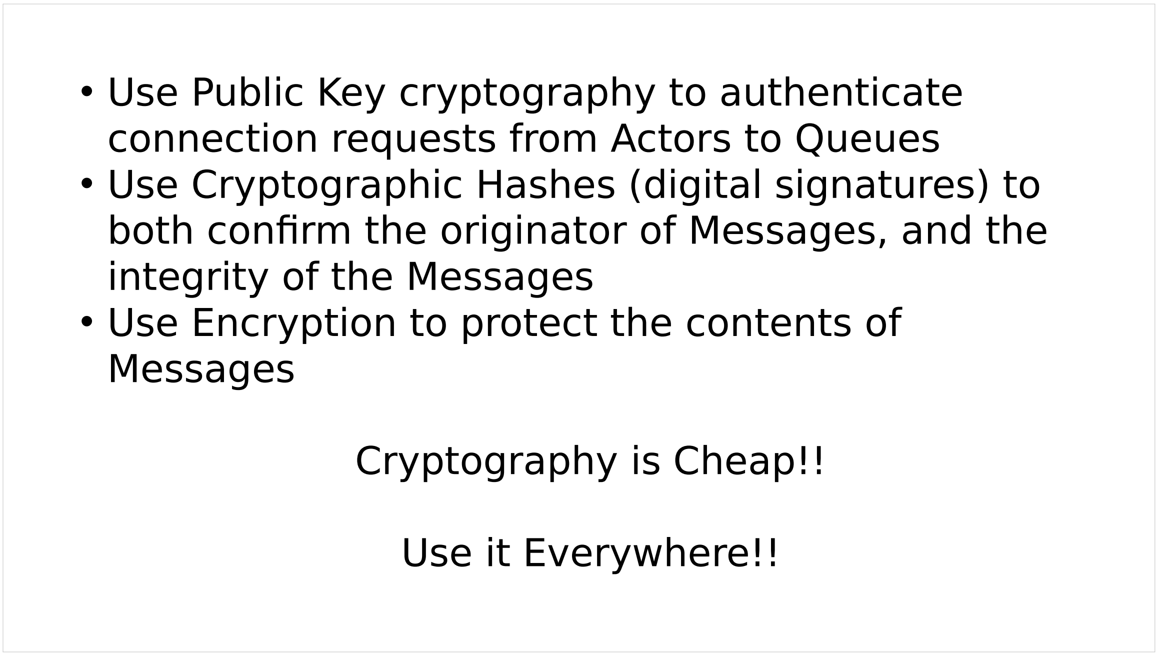

"Office Worker" Supervisor
Creates New Actors Or Terminates Actors, Based on Queue Trend
Monitors The Queue Length
arrivals
queue
Actor
Actor
Actor (Coming Or Going)
The Next Step's Queue

So far, this is analogous to a picking list in a warehouse and a staff of pickers. Each picker takes the top request off the picking list and goes into the warehouse to retrieve the required item. How do you determine if you have enough warehouse staff? Simple, watch the pick list. If it gets progressively longer, you don't have enough staff. The pick list will continue to accumulate requests until you can hire enough staff to stabilize or reduce the list. Simple. Nothing lost and you were able to monitor a gradually changing situation. What does it mean when the pick list is empty and you have warehouse staff standing around drinking coffee? Time to redeploy them to another part of the company. With our Actors, we'll just ask them to terminate themselves. OK, let's connect a couple of dots here. I said the Supervisor Actor monitored the workload and started or stopped Office Worker or Supply Clerk Actors based on this workload. The Supervisor monitors the workload by watching the input message queue to the Actor cluster. The Supervisor sees the queue increasing in size long before the situation becomes critical. If you monitor workload by measuring the CPU utilization of the server, or the I/O throughput, or the memory utilization, you are going to see sudden spikes that may kill your server or cause your Actors to fail before you can react. The queue gives you a much better and safer means of monitoring.



1. Actor A is processing the “red” message.
2. The next Actor to request a message will be given “Next Msg”
3. If Actor A doesn’t confirm to the Queue (within X seconds) that it sent the message on to the Next Step Queue, the red will become the “Next Msg” and will be handed back out

What happens if an Actor dies before completing the processing of a message? Is the message lost? [What do you mean you lost my deposit!!!!?] Or do you devise a scheme where messages have to be retained by their originating Actor until the processing Actor can confirm completion, like a TCP transmission? With this type of complexity, you are setting yourself up for failure. Instead, “use the queue, Luke”. When an Actor instance takes a message out of the queue, don’t destroy the message, just put it in a “being worked” status. If the processing Actor doesn’t tell the queue it has completed the requested action within a certain timeframe, the queue can just move the message back into the “ready” status and give it to the next member of the cluster that asks for a message. The message isn’t removed completely from the queue until it’s been completed and acknowledged. Worried about a whole queue failing? Simple – send all the messages to two redundant queues. Let the two queues keep themselves in relative synchronization. If an Actor finds it can’t talk to its primary message queue, it just connects to the redundant secondary queue and keeps on processing. You won’t lose any messages but you might re-process several messages. If the processing isn’t idempotent, then create a unique key for each message and keep track of keys. Discard any duplicate messages or results. This avoids [What do you mean you deducted my one withdrawal twice!!!!]



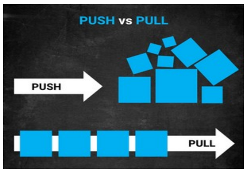
- Use Public Key cryptography to authenticate connection requests from Actors to Queues
- Use Cryptographic Hashes (digital signatures) to both confirm the originator of Messages, and the integrity of the Messages
- Use Encryption to protect the contents of Messages

Cryptography is Cheap!!

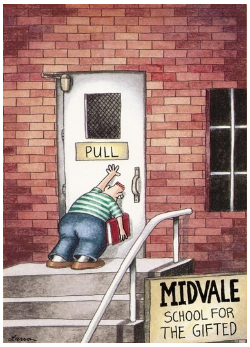
Use it Everywhere!!

Finally, queues eliminate active load balancers and the need to register and deregister Actors. As Supervisors create new Actors, the Actors start pulling the top message out of the queue. We have already discussed what happens if an Actor dies or terminates. Again, no need to deregister. If you have security concerns [and you should!!!], use public keys to authenticate connection requests to a queue, use cryptographic hashes (digital signatures) to confirm both the originator of a message and the integrity of a message, and use cryptographic encryption to protect the contents of a message. Cryptography is cheap. Use it everywhere!

The Four Ingredients in the “Secret Sauce” for Developing Infinitely Scaling High Performance Systems

3) 

Now, onto the third ingredient…

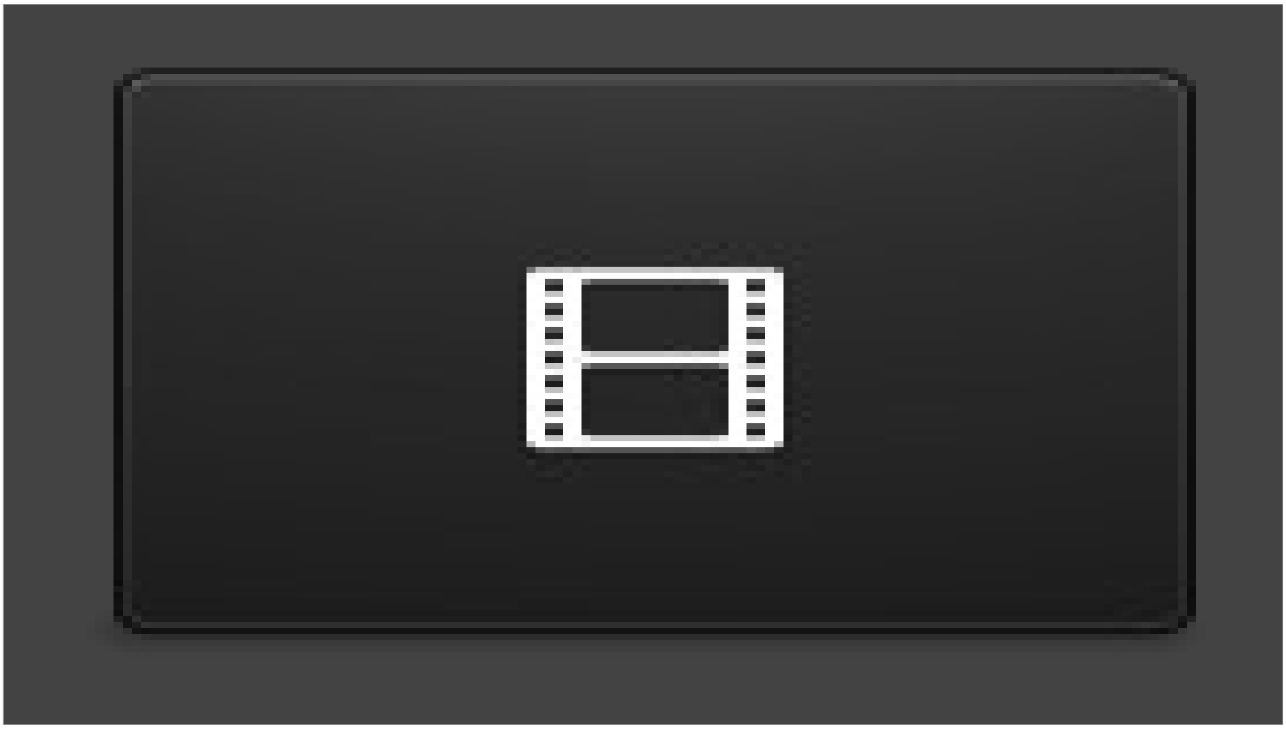


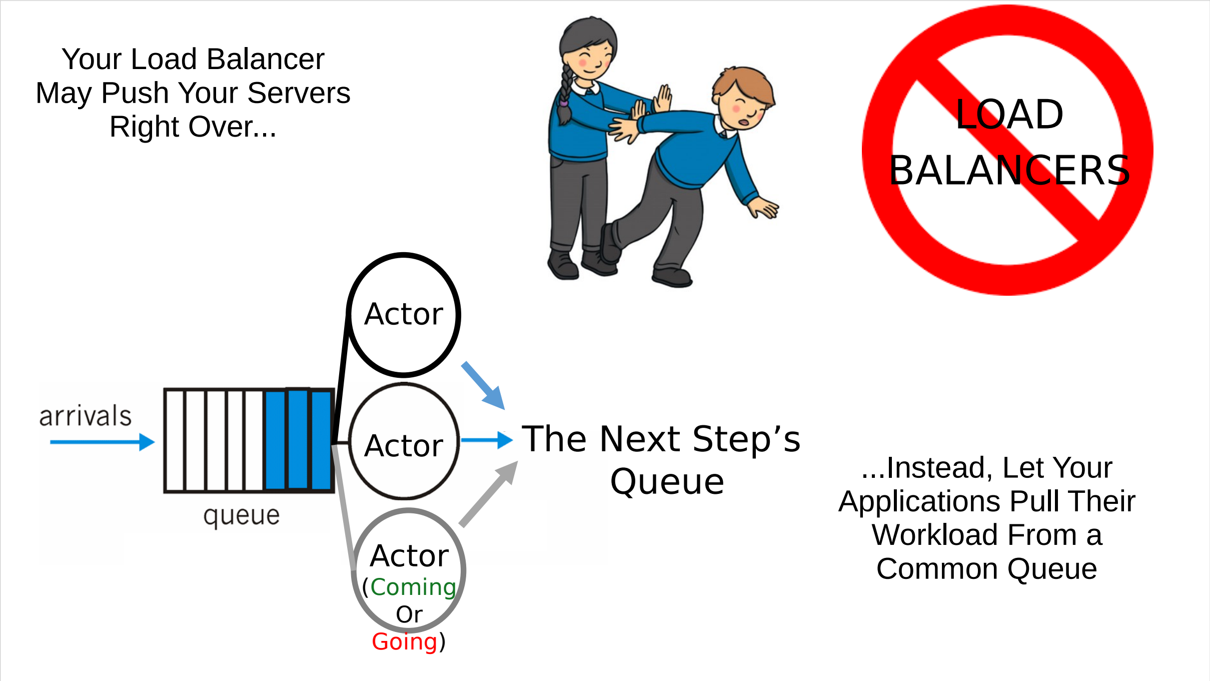
Fudd's First Law of Opposition: "If you push something hard enough, it WILL fall over."
- Firesign Theater, 1971

We'll be saying a big hello to all intelligent lifeforms everywhere and to everyone else out there, the secret is to ~~bang the rocks together~~ pull, don't push, guys."
- Douglas Adams, Hitchhiker's Guide to the Galaxy

Here's where Gary Larson's Farside cartoon taught me the key to success. Pull things, don't push them! A corollary of this is Fudd's First Law of Opposition – If you push something hard enough, it will fall over. So what happens when you push too many requests to a server? It crashes.

Here's Lucille Ball to graphically demonstrate to us why you don't want to push…





Your Load Balancer
May Push Your Servers
Right Over...

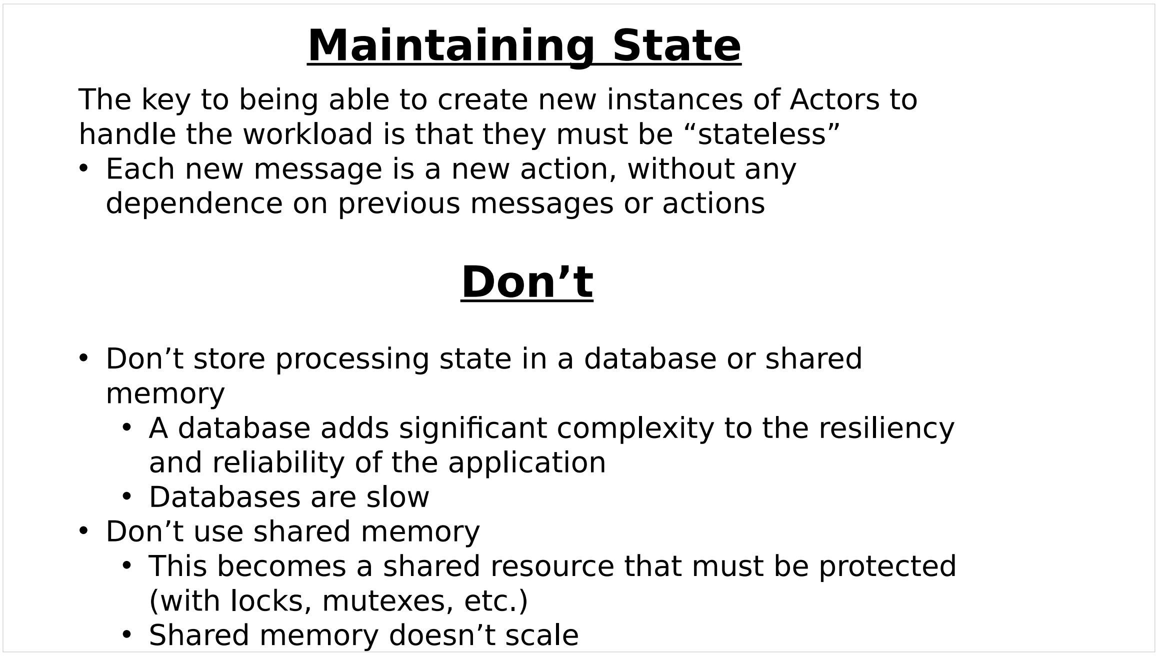
LOAD
BALANCERS

...Instead, Let Your
Applications Pull Their
Workload From a
Common Queue

A load balancer is a bully, pushing your servers over. Instead, put your requests in a queue and let your servers pull the requests. Is the queue backing up or are processing time too slow? Just spin up more server processes and attach them to the queue.

The keys to building a successful, infinitely scaling application is to:

One, Use the modified Actor Model patterns of the Office Worker, the Supply Clerk, and the Supervisor. Connect them together with Queues, and always be polit and Pull, don’t Push ;-)



Maintaining State

The key to being able to create new instances of Actors to handle the workload is that they must be “stateless”

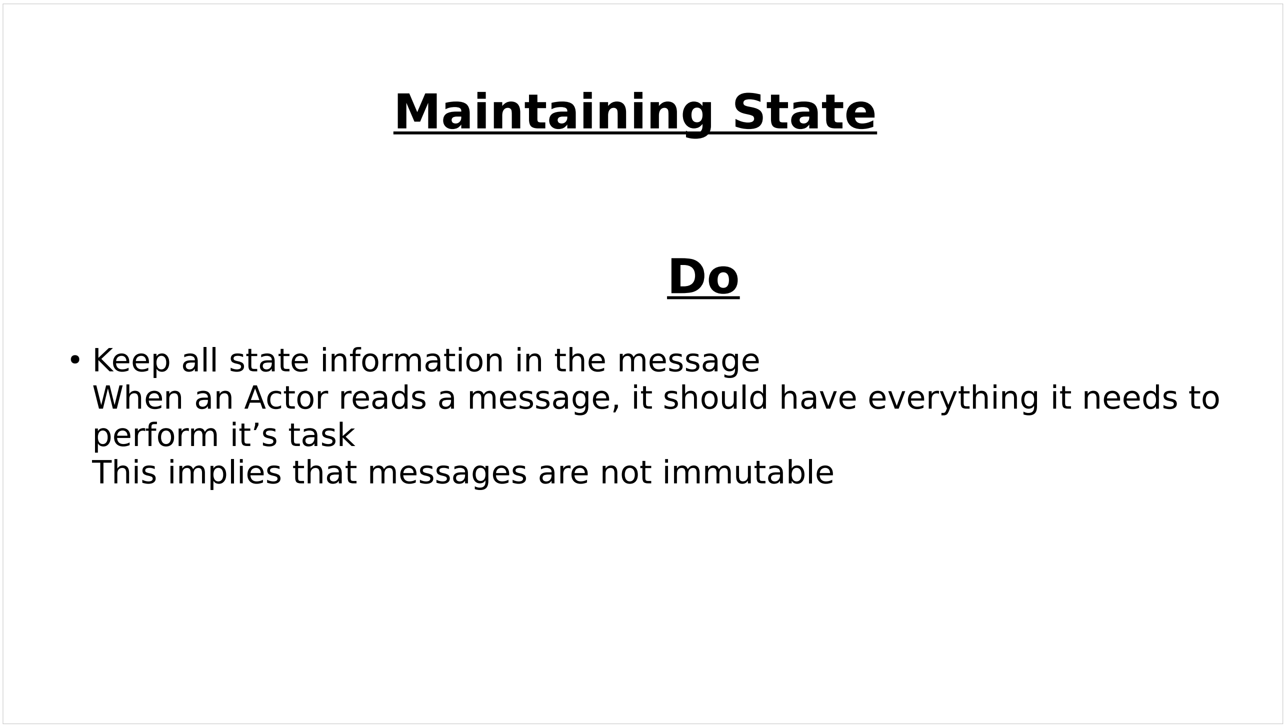
- Each new message is a new action, without any dependence on previous messages or actions

Don’t

- Don’t store processing state in a database or shared memory
  - A database adds significant complexity to the resiliency and reliability of the application
  - Databases are slow
- Don’t use shared memory
  - This becomes a shared resource that must be protected (with locks, mutexes, etc.)
  - Shared memory doesn’t scale

I just want to touch on two remaining aspects of a successful large scale program; first, how to maintain state and; second how to handle long-running processes.

I’ve talked about Actor clusters and multiple Actors pulling their work assignments out of a common queue. This only works if the Actors are, for the most part, “stateless”. Each Actor sees each new message as a new task, with no knowledge of previous tasks.

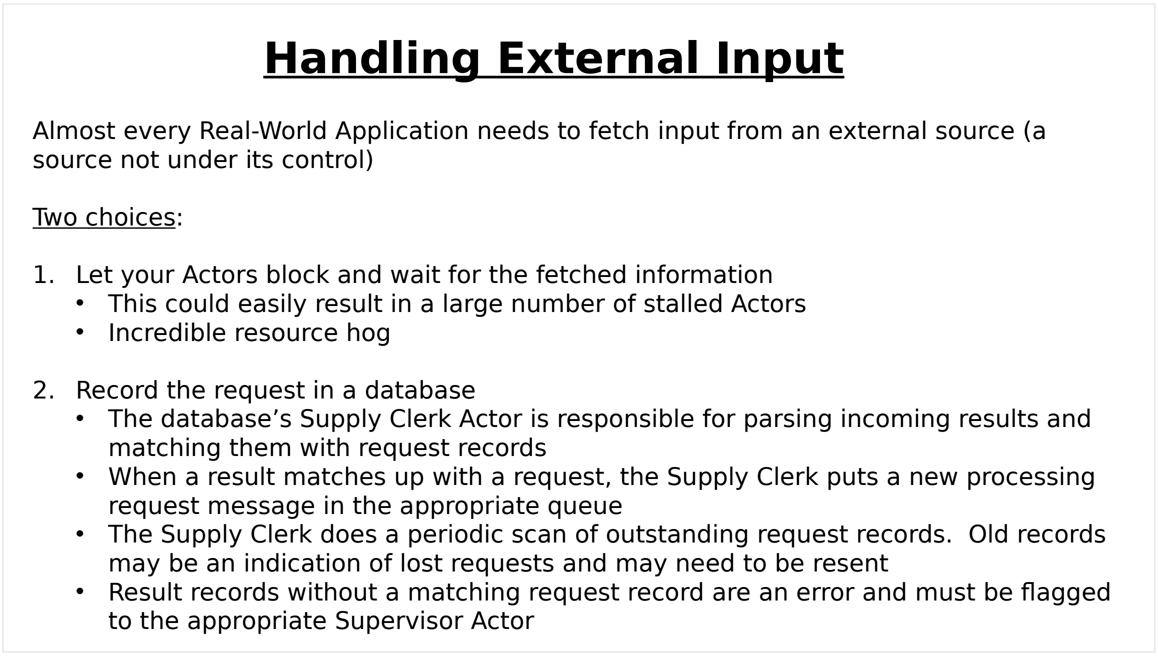


Maintaining State

Do

- Keep all state information in the message
  When an Actor reads a message, it should have everything it needs to perform it’s task
  This implies that messages are not immutable

In an Actor model system, put the state in the message. In other words, when an Actor puts up a new message, the message holds all the information required to accomplish the Actor’s process. The Actor doesn’t have to go fetch the process state from a data base or from shared memory. This also implies that messages are not immutable. Each Actor, as it completes its processing and sends the message on to the next step in the overall process must include everything that the next Actor will require to complete its task.



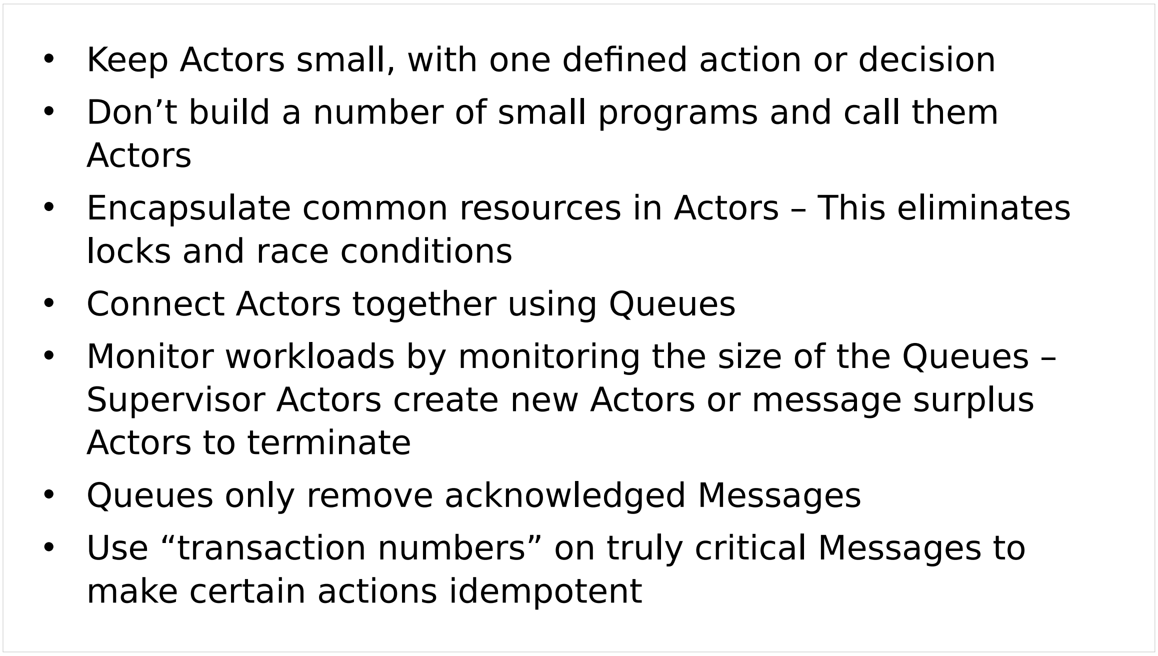
Handling External Input

Almost every Real-World Application needs to fetch input from an external source (a source not under its control)

Two choices:

1. Let your Actors block and wait for the fetched information
   - This could easily result in a large number of stalled Actors
   - Incredible resource hog
2. Record the request in a database
   - The database's Supply Clerk Actor is responsible for parsing incoming results and matching them with request records
   - When a result matches up with a request, the Supply Clerk puts a new processing request message in the appropriate queue
   - The Supply Clerk does a periodic scan of outstanding request records. Old records may be an indication of lost requests and may need to be resent
   - Result records without a matching request record are an error and must be flagged to the appropriate Supervisor Actor

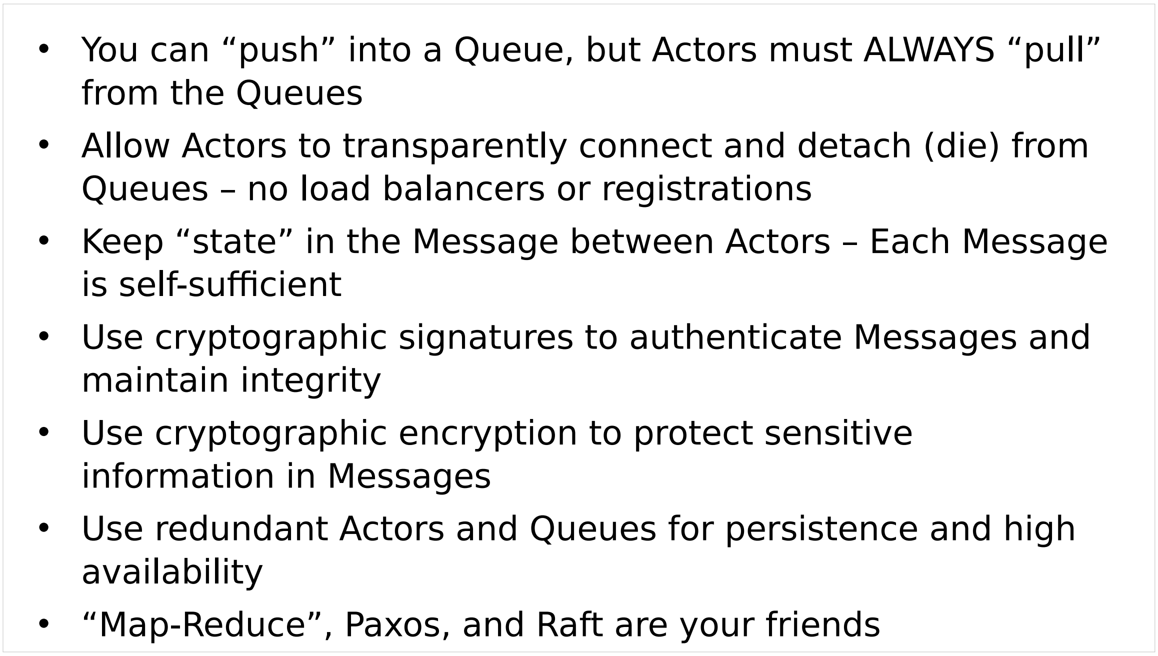
very few meaningful business applications are completely self-contained. In most cases, there is a need to access information that's external to our application. If it's a Federal Health Insurance Exchange, we may need to confirm information from an enrollee's Federal Tax return. The IRS is great about this. They collect all your information requests, run a batch job at night and send you back the answers the next day. So it could be as long as 24 hours before your request is satisfied. Now, you could just create a new Actor for each outstanding request. As the answers come back, each Actor could then complete their process and go on to the next job. But if we are enrolling 1 million people per day, that's a lot of stalled Actors standing around waiting. Instead of creating stalled Actors, let's use our Supply Clerk-type Actors. As the IRS Supply Clerk Actor to put the original message into a database. When the IRS results come in, ask the IRS Supply Clerk to retrieve the original message. Add the IRS results to the original message and put it into the queue of the cluster handling the next step in the enrollment process. You also have a nice error detection process. If a message stays in the database more than a day, then the IRS probably lost the original request and we can send it again. By the same token, if the IRS sends us back a result for which we don't have an original message stored, then we have an internal problem and we immediately raise a red flag.



- Keep Actors small, with one defined action or decision
- Don’t build a number of small programs and call them Actors
- Encapsulate common resources in Actors – This eliminates locks and race conditions
- Connect Actors together using Queues
- Monitor workloads by monitoring the size of the Queues – Supervisor Actors create new Actors or message surplus Actors to terminate
- Queues only remove acknowledged Messages
- Use “transaction numbers” on truly critical Messages to make certain actions idempotent

OK, at this point, I think you know enough to go out and architect your own large scale web application.

In Review...



- You can “push” into a Queue, but Actors must ALWAYS “pull” from the Queues
- Allow Actors to transparently connect and detach (die) from Queues – no load balancers or registrations
- Keep “state” in the Message between Actors – Each Message is self-sufficient
- Use cryptographic signatures to authenticate Messages and maintain integrity
- Use cryptographic encryption to protect sensitive information in Messages
- Use redundant Actors and Queues for persistence and high availability
- “Map-Reduce”, Paxos, and Raft are your friends

First, you can “push” into a Queue, but Actors must ALWAYS “pull” from the Queues

Second, allow Actors to transparently connect and detach (die) from Queues – no load balancers or registrations

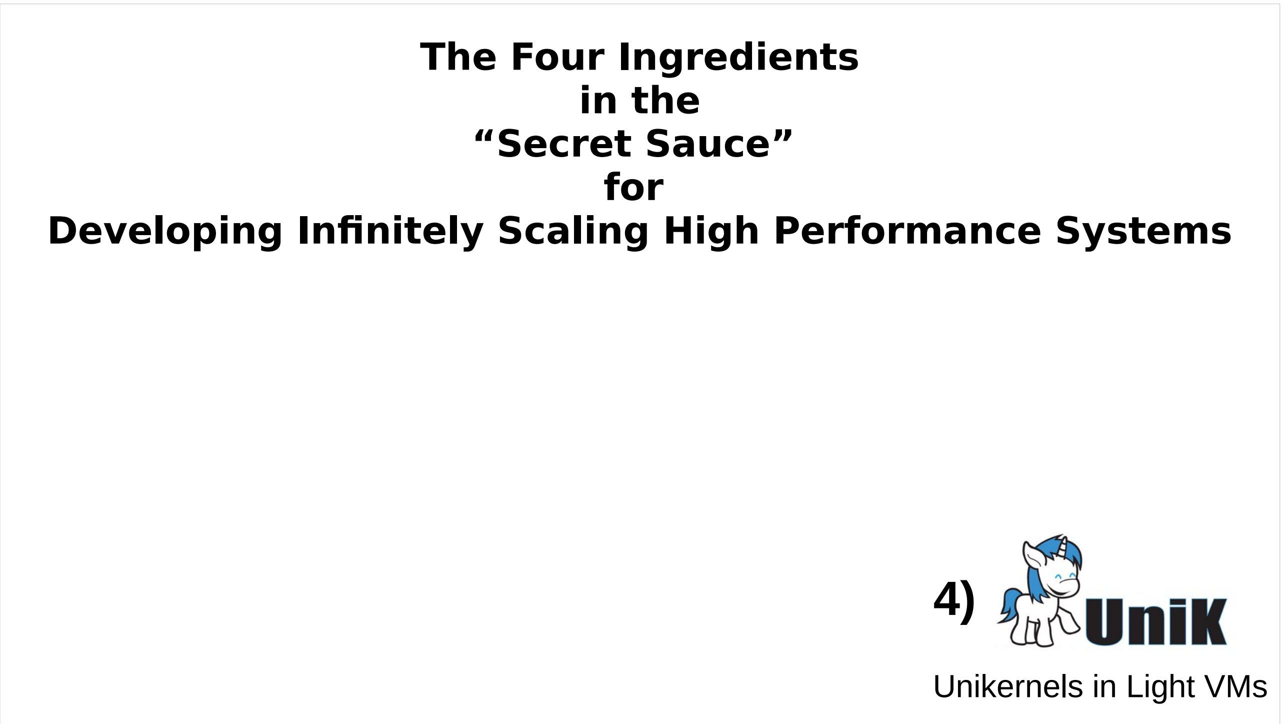
Third, keep “state” in the Message between Actors – Each Message is self-sufficient

Fourth, use cryptographic signatures to authenticate Messages and maintain integrity

Fifth, use cryptographic encryption to protect sensitive information in Messages

Sixth, use redundant Actors and Queues for persistence and high availability, and finally

Seventh, “Map-Reduce”, Paxos, and Raft are your friends



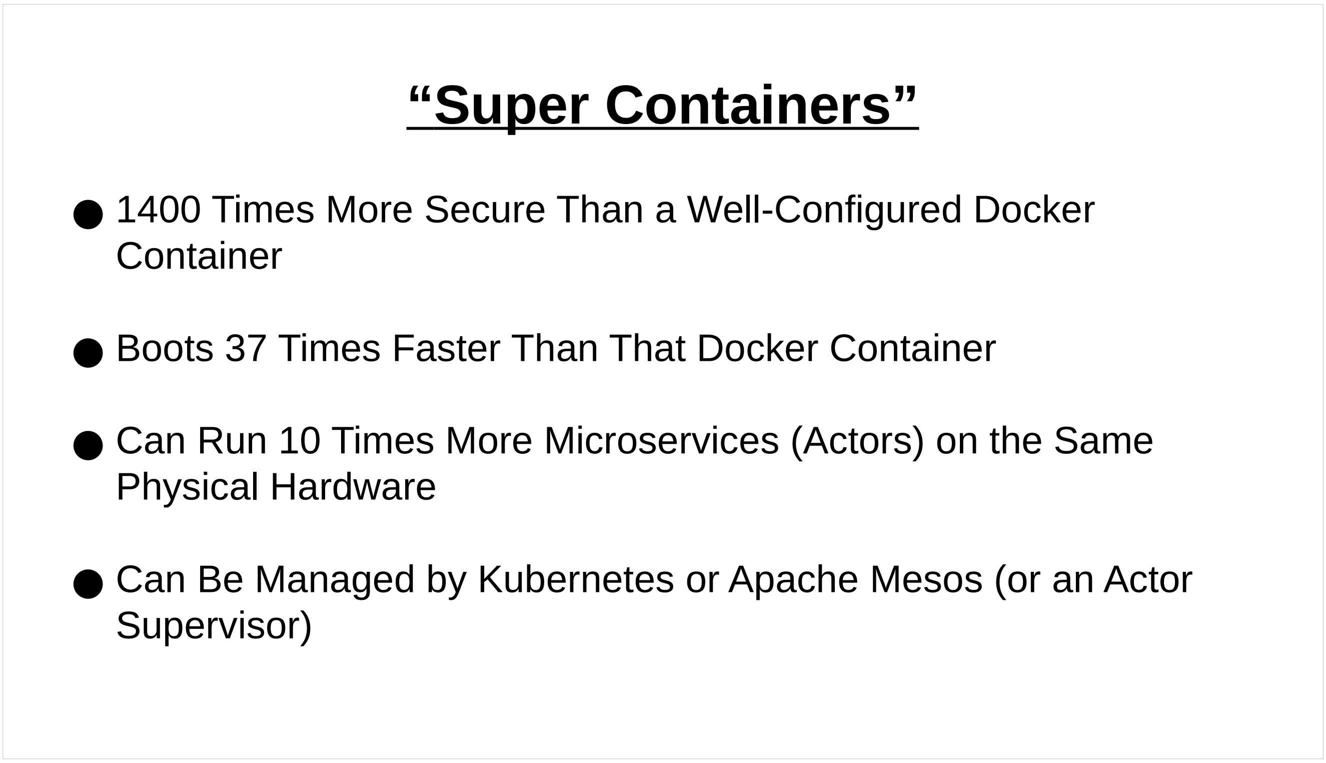
The Four Ingredients in the “Secret Sauce” for Developing Infinitely Scaling High Performance Systems

4) UniK

Unikernels in Light VMs

So at this point, you have thousands of Actors, with thousands more coming and going as Supervisors balance out the queues. Do we need Googleplex data centers? Even with Docker and Kubernetes, we may still talking a large number of physical servers.

The solution, and final ingredient in our Secret Sauce are Super Containers!

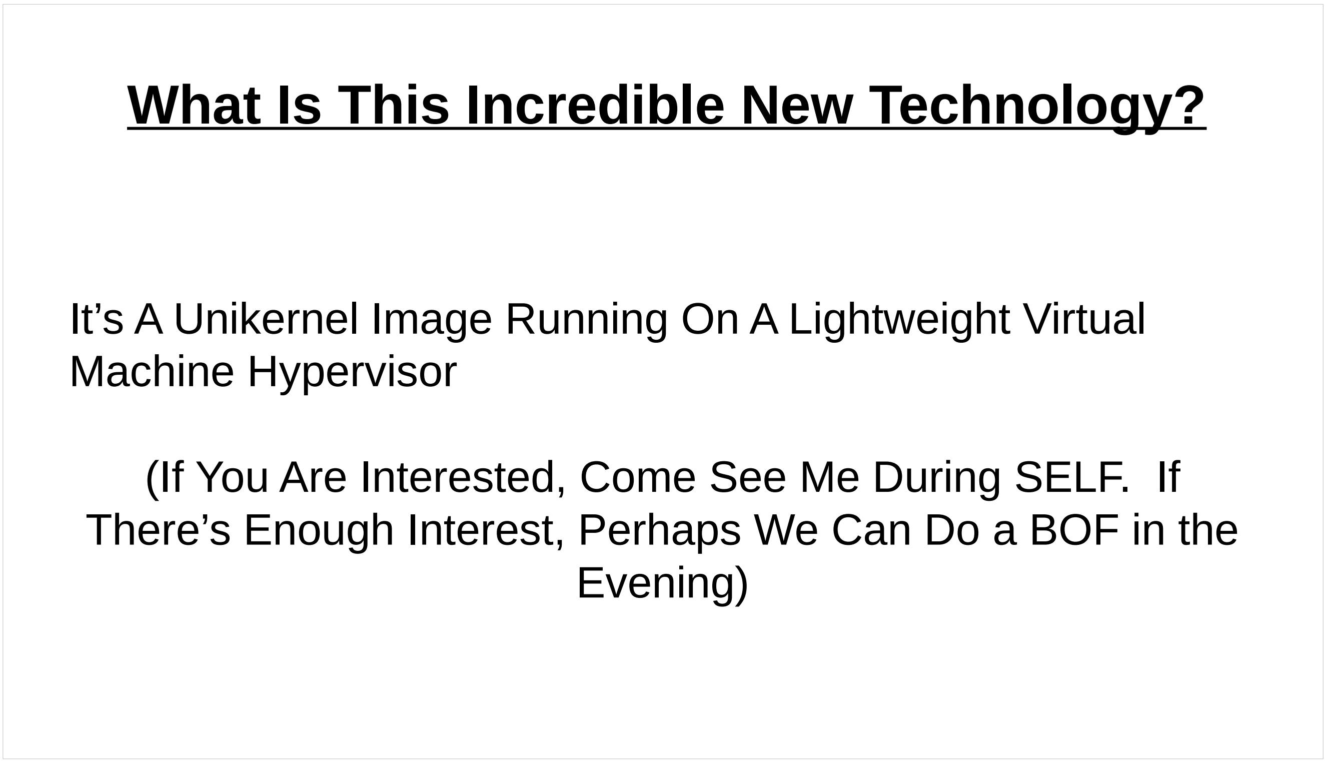


“Super Containers”

- 1400 Times More Secure Than a Well-Configured Docker Container
- Boots 37 Times Faster Than That Docker Container
- Can Run 10 Times More Microservices (Actors) on the Same Physical Hardware
- Can Be Managed by Kubernetes or Apache Mesos (or an Actor Supervisor)

Super Containers - That’s not their real name, but what else would you call a container that’s:

- 1400 Times More Secure Than A Well-configured Docker Container
- Boots 37 Times Faster Than That Docker Container
- Can Run 10 Times More Microservices On The Same Physical Hardware
- Can Be Managed By Kubernetes Or Apache Mesos



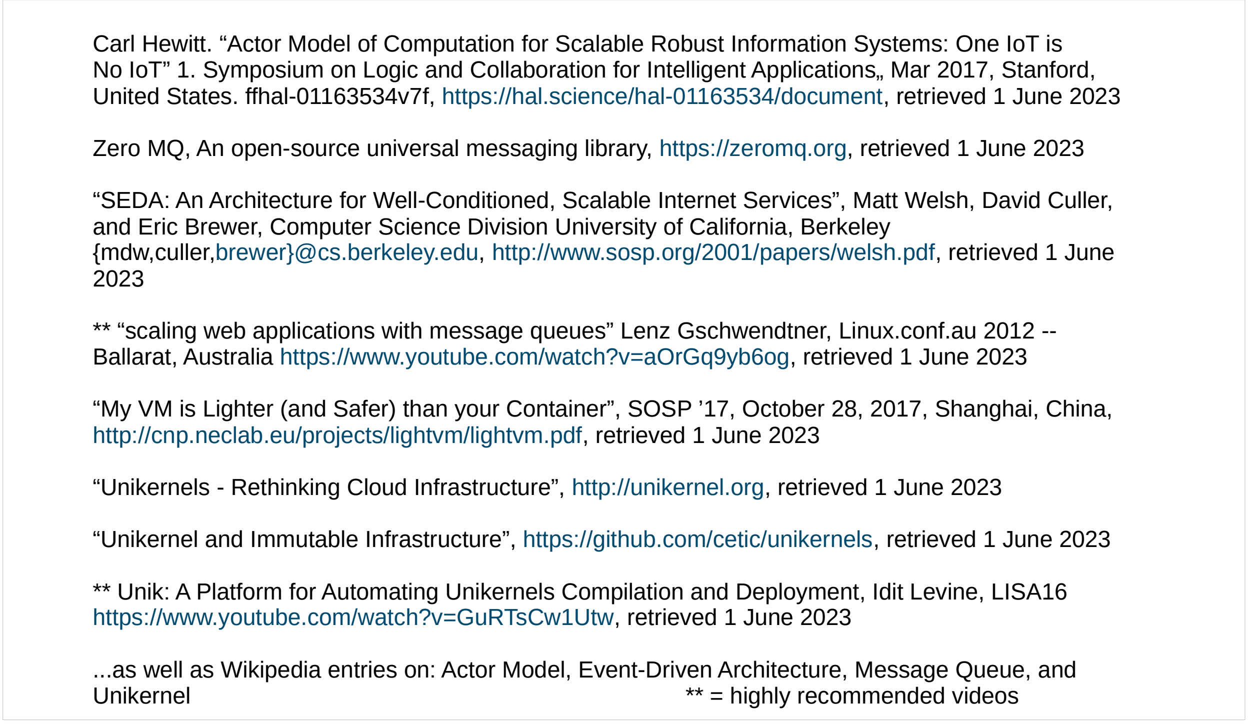
What Is This Incredible New Technology?

It’s A Unikernel Image Running On A Lightweight Virtual Machine Hypervisor

(If You Are Interested, Come See Me During SELF. If There’s Enough Interest, Perhaps We Can Do a BOF in the Evening)

What Is This Incredible New Technology?

It’s A Unikernel Image Running On A Lightweight Virtual Machine Hypervisor



Carl Hewitt. “Actor Model of Computation for Scalable Robust Information Systems: One IoT is No IoT” 1. Symposium on Logic and Collaboration for Intelligent Applications„ Mar 2017, Stanford, United States. ffhal-01163534v7f, https://hal.science/hal-01163534/document, retrieved 1 June 2023

Zero MQ, An open-source universal messaging library, https://zeromq.org, retrieved 1 June 2023

“SEDA: An Architecture for Well-Conditioned, Scalable Internet Services”, Matt Welsh, David Culler, and Eric Brewer, Computer Science Division University of California, Berkeley {mdw,culler,brewer}@cs.berkeley.edu, http://www.sosp.org/2001/papers/welsh.pdf, retrieved 1 June 2023

** “scaling web applications with message queues” Lenz Gschwendtner, Linux.conf.au 2012 -- Ballarat, Australia https://www.youtube.com/watch?v=aOrGq9yb6og, retrieved 1 June 2023

“My VM is Lighter (and Safer) than your Container”, SOSP ’17, October 28, 2017, Shanghai, China, http://cnp.neclab.eu/projects/lightvm/lightvm.pdf, retrieved 1 June 2023

“Unikernels - Rethinking Cloud Infrastructure”, http://unikernel.org, retrieved 1 June 2023

“Unikernel and Immutable Infrastructure”, https://github.com/cetic/unikernels, retrieved 1 June 2023

** Unik: A Platform for Automating Unikernels Compilation and Deployment, Idit Levine, LISA16 https://www.youtube.com/watch?v=GuRTsCw1Utw, retrieved 1 June 2023

...as well as Wikipedia entries on: Actor Model, Event-Driven Architecture, Message Queue, and Unikernel

** = highly recommended videos

Here a few key links to further information of the Actor Model, using queues for performance control, and why you should always pull, not push. I’ve flagged two YouTube videos that are particularly applicable

**Copies of the slides and the talking points may be downloaded from the
Formularity website:**

https://formularity.com

All illustrations and videos were used under the Fair Use Doctrine of US Copyright Law, for the purpose of education.

OK, so did you learn anything new? Anything you didn't already know before you came in here today? See, I told you so.

Thank you for your time. A copy of the slides with my speaker script is on the Formularity website.

I'll take any questions now.