

Learning the Three Types of Microservices

<http://g.mamund.com/at-newsletter-signup>

CodeMash 2019

Mike Amundsen

@mamund

API Academy



Mike Amundsen
@mamund

training.amundsen.com

Amundsen Training

Home  

Learning with Mike Amundsen



[Technical](#)



[Transformation](#)



[Teams](#)



[Inspiration](#)

www.apiacademy.co



[Explore](#)

[Resources](#)

[Events](#)

[Podcast](#)

[About](#)

[Subscribe](#)



Get updates to your email



Subscribe

API STRATEGY

Microservices, APIs and Innovation: The Power of APIs

Explore the role APIs play in empowering teams and enabling organizations to innovate.

<http://g.mamund.com/msabook>

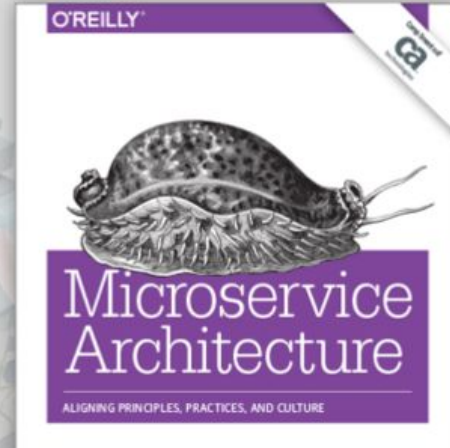


Microservice Architecture: Aligning Principles, Practices, and Culture

Microservices is the next evolution in software architecture designed to help organizations embrace continual change in the digital economy. But how do you design and apply an effective microservice architecture?

This new book from O'Reilly provides comprehensive guidance through seven valuable chapters that give you a deep-dive into:

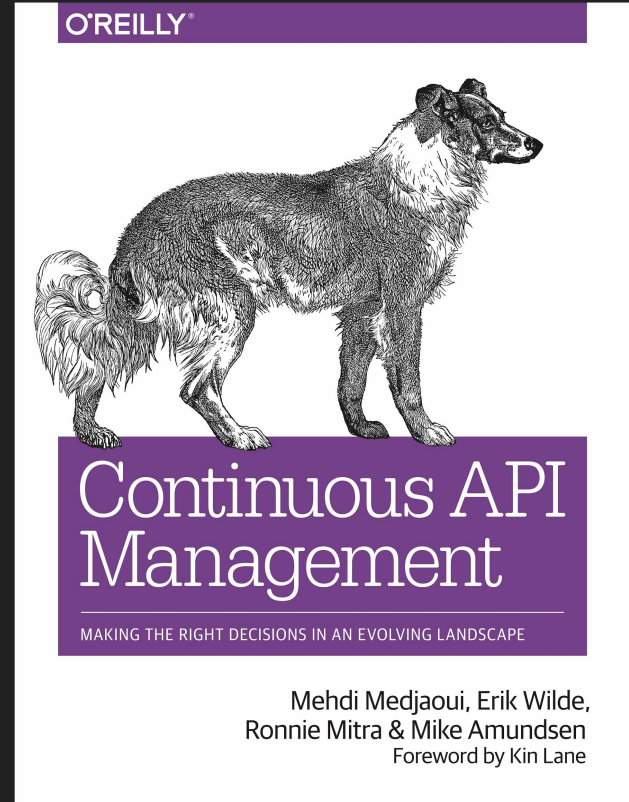
- The benefits and principles of microservices
- A design-based approach to microservice architecture
- Lessons for applying microservices in practice



<http://g.mamund.com/cambook>

"A reusable guide to the technology, business, and politics of doing APIs at scale within the enterprise."

-- Kin Lane, API Evangelist



Overview

- Programming the Network
- Microservices
- Three Types of Microservice Components
- Nygard's Stability Patterns
- Applying Nygard to Microservices
- But Wait, There's More...

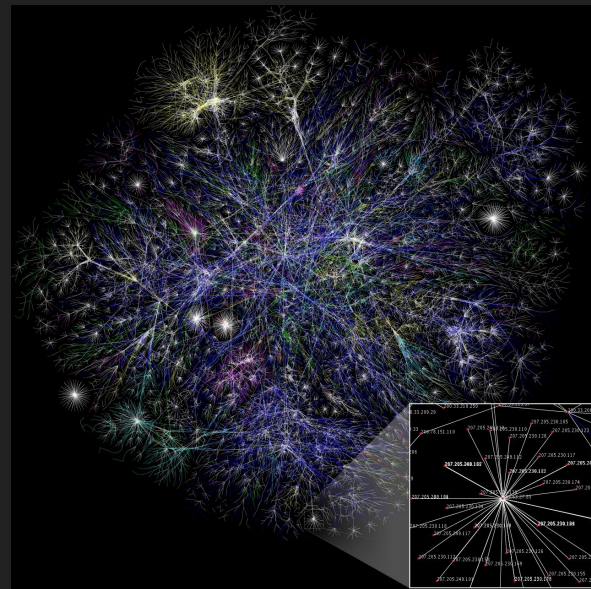
Programming the Network



Traveling

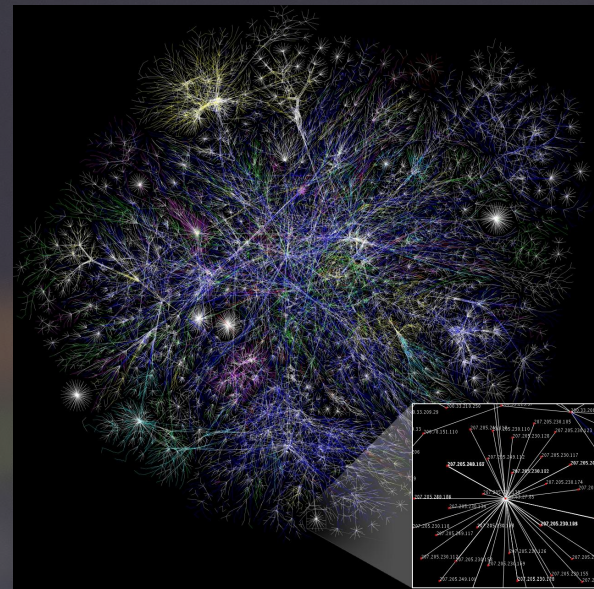


Traveling the Network





Programming the Network



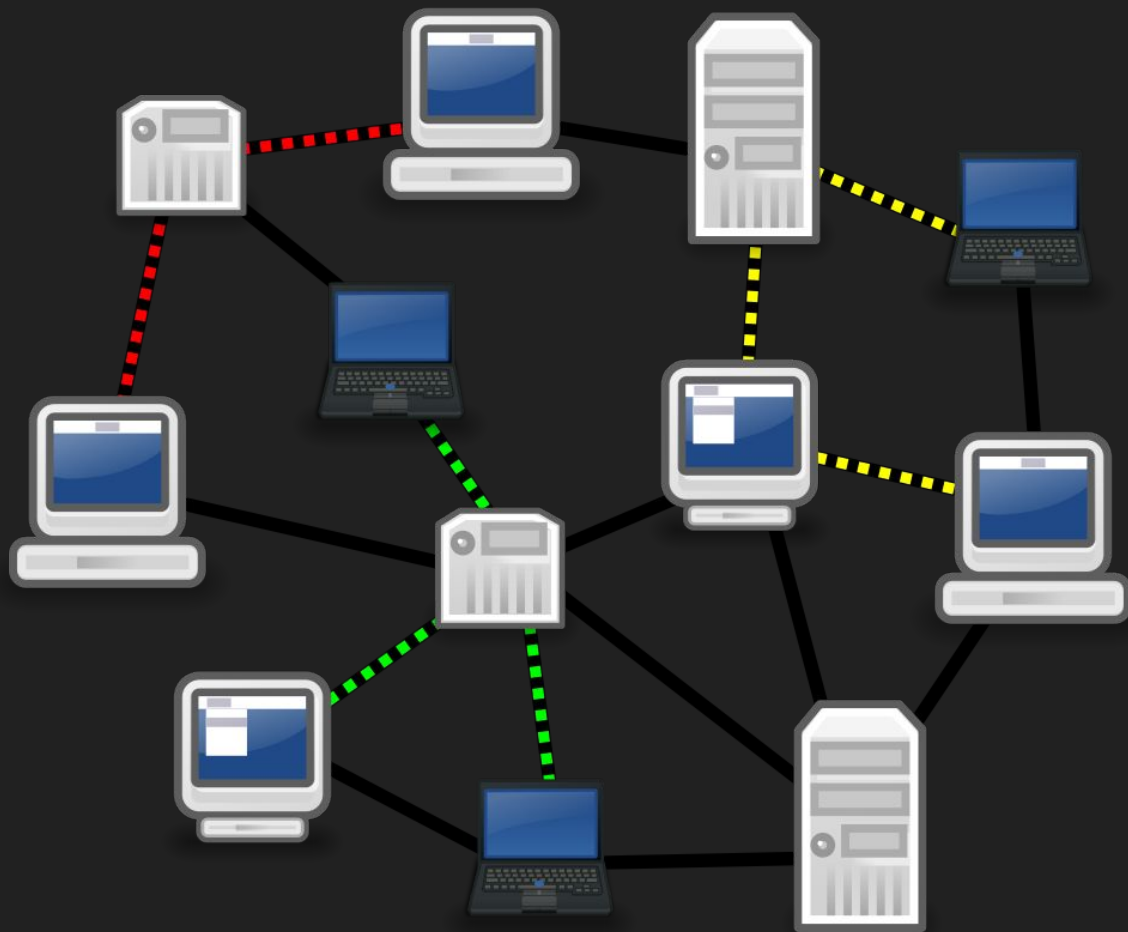
RedBoot(tm) bootstrap and debug environment [RAM]
(Panasonic Avionics Corporation) release, version ("560328-212" v "1.07" b "0126"
- built 15:35:59, May 22 2013

Platform: SM-02 (I386)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x000a0000, 0x00100000-0x01000000 available
Current Boot Count is 0

verifying MBR... Fix MBR:
Partition 0: already exists
Partition 1: already exists
Partition 2: already exists
Partition 3: already exists

verifying image... OK.
== Executing kernel in 5 seconds - enter ^C to abort
Load Address 0x00000000
Image length 0x00e2f5d5
Loading kernel binary...
Read image signature... 1f 8b
Decompressing image...



Programming the Network

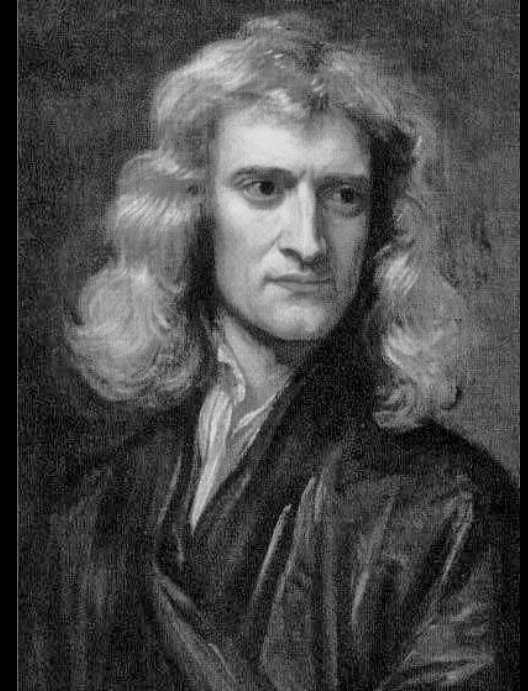
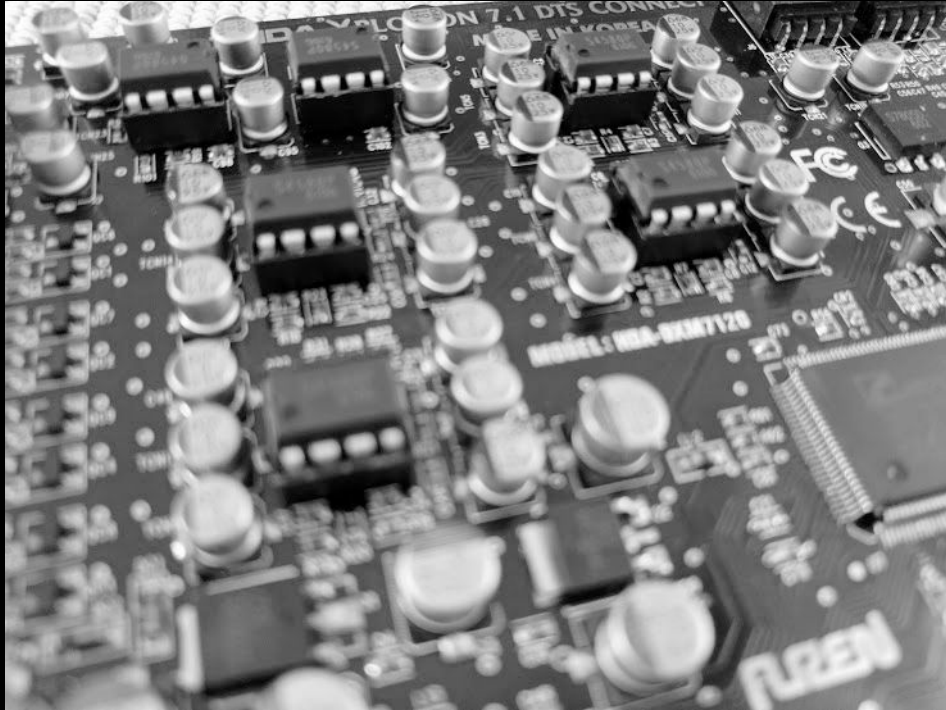
"There is no simultaneity at a distance."

-- Pat Helland (2005)



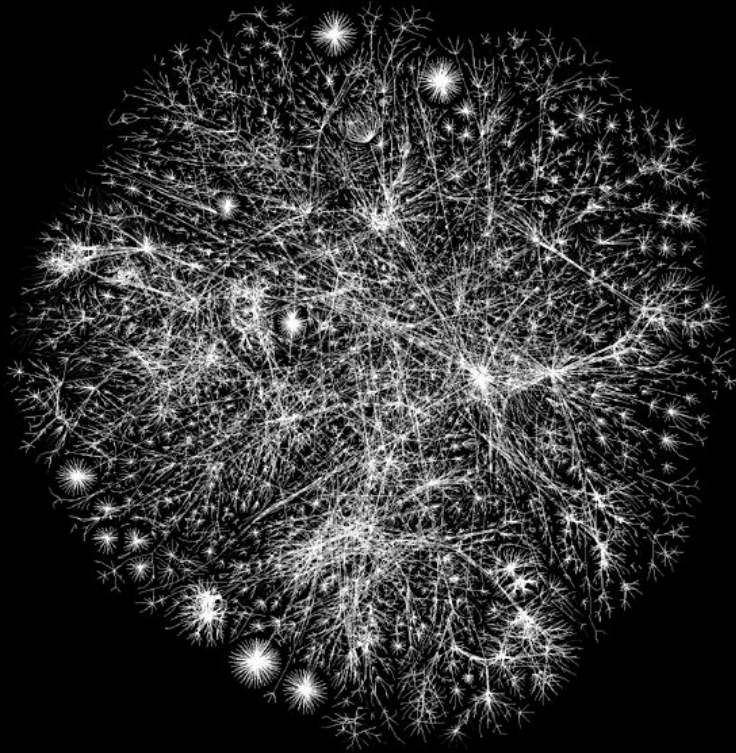
Pat Helland

Newton rules the "inside"



Sir Isaac Newton

Einstein rules the "outside"



Albert Einstein

Programming the Network

There is no simultaneity at a distance!

- Similar to the speed of light bounding information
- By the time you see a distant object, it may have changed!
- By the time you see a message, the data may have changed!



Pat Helland

Programming the Network

There is no simultaneity at a distance!

- Similar to the speed of light bounding information
- By the time you see a distant object, it may have changed!
- By the time you see a message, the data may have changed!

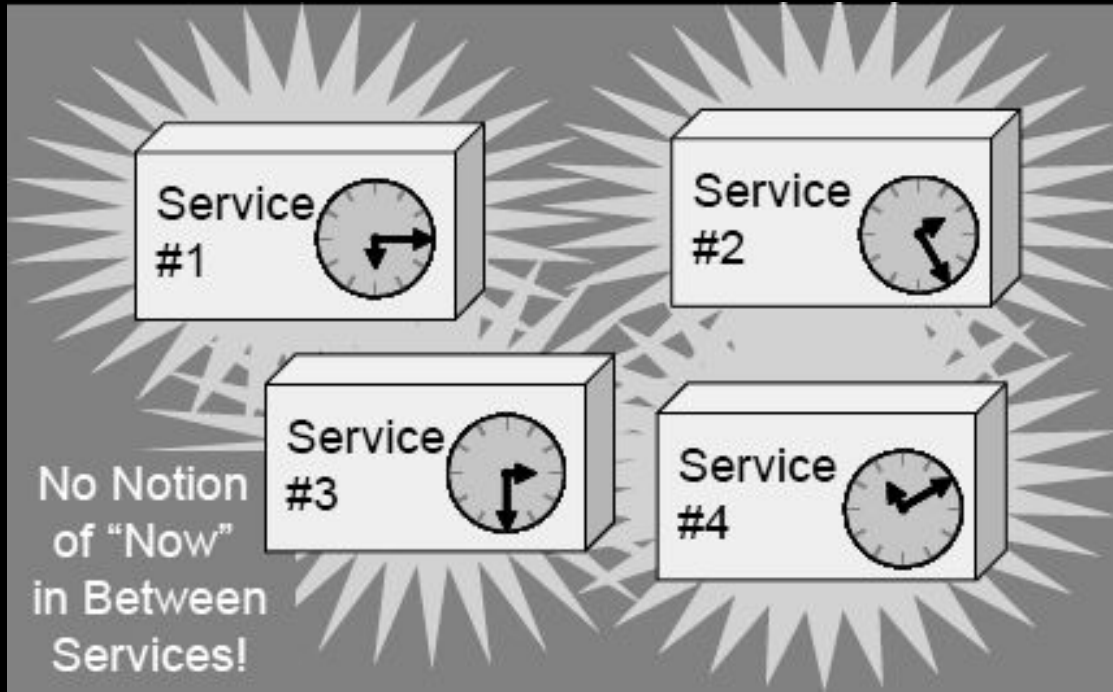


Pat Helland

Services, transactions, and locks bound simultaneity!

- Inside a transaction, things are simultaneous
- Simultaneity exists only inside a transaction!
- Simultaneity exists only inside a service!

Programming the Network



Pat Helland

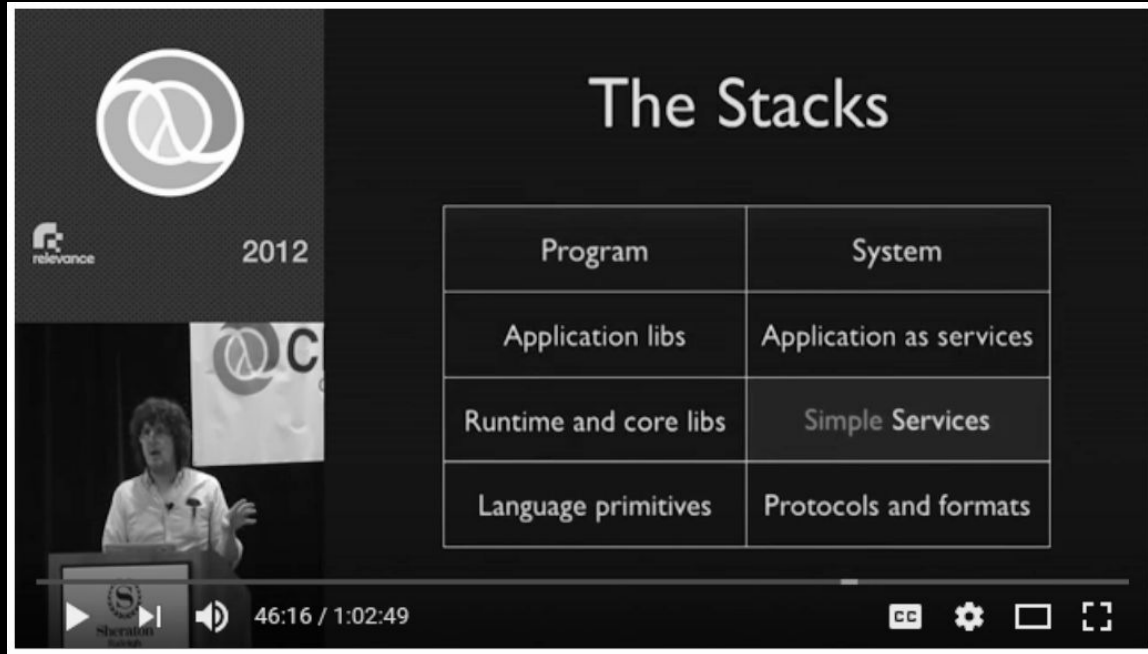
Fallacies of Distributed Computing (1994)

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.



L Peter Deutsch

The Language of the System (2012)



The screenshot shows a video player interface. In the top left corner, there is a logo consisting of two overlapping circles, the 'relevance' logo, and the year '2012'. Below this, a smaller video frame shows a woman speaking at a podium. The main content of the slide is a table titled 'The Stacks'.

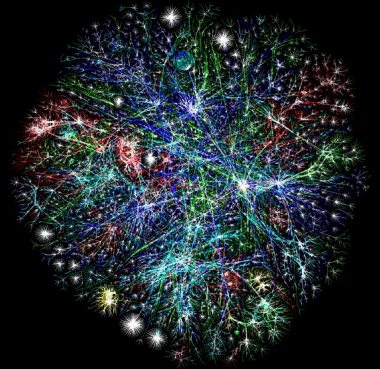
Program	System
Application libs	Application as services
Runtime and core libs	Simple Services
Language primitives	Protocols and formats

At the bottom of the video player, there is a progress bar showing '46:16 / 1:02:49' and various control icons like play, volume, and full screen.



Rich Hickey

***Programming the Network brings
new challenges***



Microservices

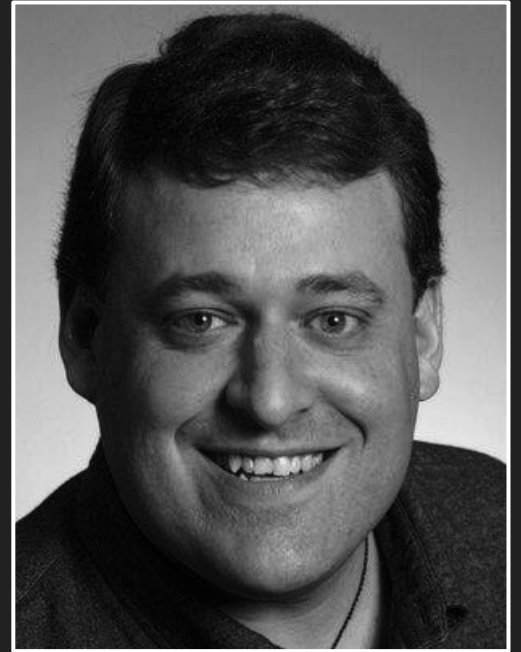
"An approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms."

-- Martin Fowler, 2014



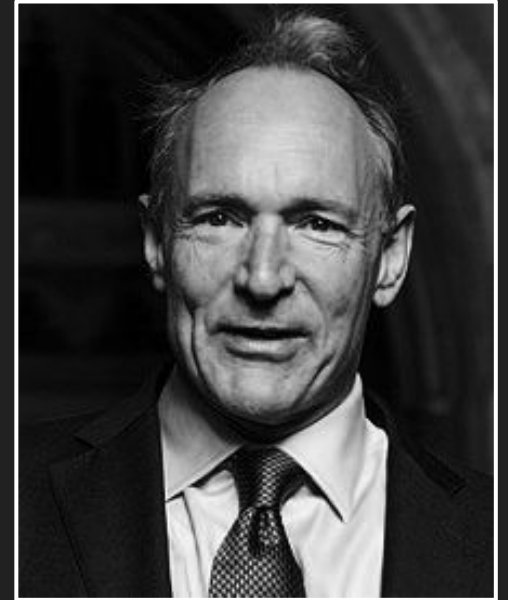
*"Emphasizes scalability of
component interactions,
generality of interfaces,
independent deployment of
components, and intermediary
components."*

-- Roy Fielding, 2000



"A universal linked information system, in which generality and portability are [most] important."

-- Tim Berners-Lee, 1989



Microservice Characteristics

- Make each program do one thing well
- Expect the output of every program to be the input of another program
- Design and build software to be tried early
- Use tools to lighten the programming task

Unix Operating Principles (1978)

- Make each program do one thing well
- Expect the output of every program to be the input of another program
- Design and build software to be tried early
- Use tools to lighten the programming task

*Loosely-coupled components
running in an
engineered system.*

Three Types of Microservices

Three Types of Microservices

- Stateless
- Persistence
- Aggregator

Stateless Microservices

Stateless Microservices

- Simple processors (converters, translators, etc.)
- No dependence on other microservices
- No local data storage (disk I/O)

The most common MSC example, but the least useful!

Stateless Microservices

- No shared state
- Easy to replace
- Easy to scale up

Ephemeral Computing

Stateless Microservices

```
// http server handling data conversions  
function conversionServer(request, response) {  
    response = convertValue(request);  
    return response;  
}
```

WARNING: NOT REAL CODE!

Persistence Microservices

Persistence Microservices

- Simple (local) storage (reads and/or writes)
- Disk I/O dependent
- Possibly VM or one-U dependent

Commonly needed MSC, not the easiest to implement.

Persistence Microservices

- System of Record/Source of Truth
- Relatively easy to scale for reads (CQRS)
- No cross-service two-phase commits (Saga)

Durable Storage

Persistence Microservices

```
function updateOrders(request, response) {  
  response = localStorage.write(request);  
  return response;  
}
```

WARNING: NOT REAL CODE!

Aggregator Microservices

Aggregator Microservices

- Depends on other ("distant") microservices
- Network dependent
- Usually Disk I/O dependence, too

The most often-needed; most challenging, too.

Aggregator Microservices

- Sequence vs. Parallel calls
- Timing is everything
- Easy to scale (should be...)

Workflow Choreography

Aggregator Microservices

```
function writeOrders(request, response) {  
  var resourceList = ["customerDB", "orderDB", "salesDB"]  
  var serviceList = gatherResources(resourceList);  
  response = serviceList(request)  
  
  return response;  
}
```

WARNING: NOT REAL CODE!

Three Types of Microservices

- Stateless (ephemeral)
- Persistence (durable)
- Aggregator (workflow)

But, what about the network?

Nygaard's Stability Patterns

“Bugs will happen. They cannot be eliminated, so they must be survived instead.”

-- Michael T. Nygard



The
Pragmatic
Programmers


Release It!

Second Edition

Design and Deploy
Production-Ready Software



Michael T. Nygard
Editor: Jay Richardson

 BETA

Nygard Stability Patterns

- **Timeout**
- **Circuit Breaker**
- **Bulkhead**
- **Steady State**
- **Fail Fast**
- **Handshaking**



"Nygard Stability Patterns" -- Timeout

"The timeout is a simple mechanism allowing you to stop waiting for an answer once you think it will not come."

-- Chapter 5.1



"Nygard Stability Patterns" -- Timeout

"The timeout is a simple mechanism allowing you to stop waiting for an answer once you think it will not come."

-- Ch 5.1

```
// set up proper shutdown
process.on('SIGTERM', function () {
  discovery.unregister(null, function(response) {
    try {
      uuidGenerator.close(function() {
        console.log('gracefully shutting down');
        process.exit(0);
      });
    } catch(e){}
  });
  setTimeout(function() {
    console.error('forcefully shutting down');
    process.exit(1);
  }, 10000);
});
```

WARNING: NOT REAL CODE!



"Nygard Stability Patterns" -- **Circuit Breaker**

"Circuit breakers are a way to automatically degrade functionality when the system is under stress."

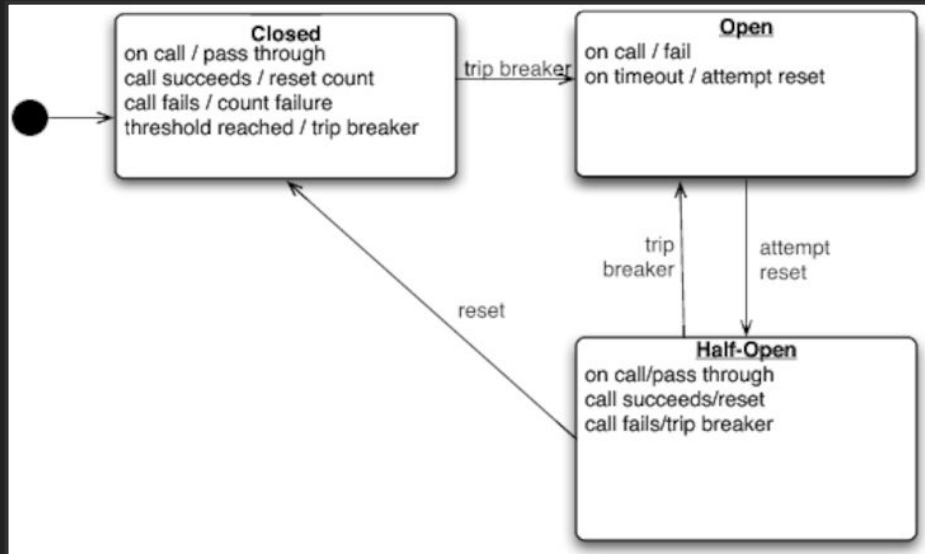
-- Chapter 5.2



"Nygard Stability Patterns" -- Circuit Breaker

"Circuit breakers are a way to automatically degrade functionality when the system is under stress."

-- Chapter 5.2



"Nygard Stability Patterns" -- Bulkhead

"The bulkhead enforces a principle of damage containment."

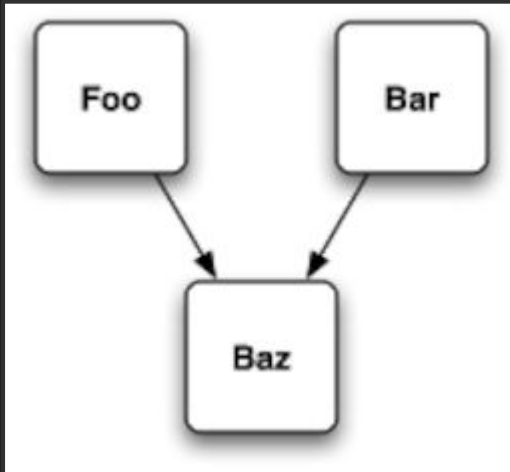
-- Chapter 5.3



"Nygard Stability Patterns" -- Bulkhead

"The bulkhead enforces a principle of damage containment."

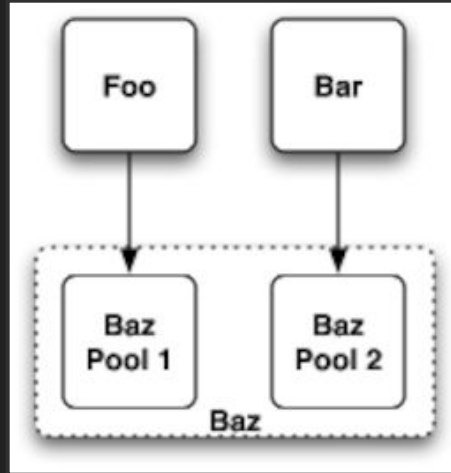
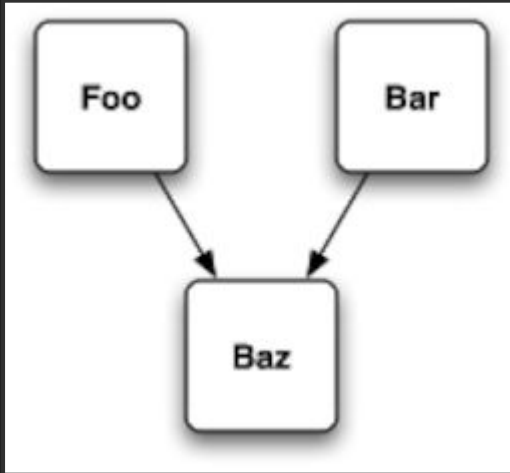
-- Chapter 5.3



"Nygard Stability Patterns" -- Bulkhead

"The bulkhead enforces a principle of damage containment."

-- Chapter 5.3



"Nygard Stability Patterns" -- **Steady State**

"The system should be able to run indefinitely without human intervention."

-- Chapter 5.4

- Avoid fiddling
- Purge data w/ app logic
- Limit caching
- Roll the logs



"Nygard Stability Patterns" -- **Steady State**

"The system should be able to run indefinitely without human intervention."

-- Chapter 5.4

- Avoid fiddling
- Purge data w/ app logic
- Limit caching
- Roll the logs



"Nygard Stability Patterns" -- **Fail Fast**

"If the system can determine in advance that it will fail at an operation, it's always better to fail fast."

-- Chapter 5.5



"Nygard Stability Patterns" -- Fail Fast

"If the system can determine in advance that it will fail at an operation, it's always better to fail fast."

-- Chapter 5.5

```
function bookOrders(orderList, timeBudget) {
  var status = false;
  var resources = ["customerdata", "orderdata", "salesdata"];
  setTimeout(function(resources) {
    var status = confirmResourceAvailability(resources);
    if(status===true && timeBudget>500) {
      try {
        status = writeOrders(orderList,resources);
      }
      catch (ex) {
        error("failed to write orders : {errorcode}",ex);
      }
    }
    else {
      error("failed to acquire resources : FAILFAST");
    }
  }, timeBudget);
}
```

WARNING: NOT REAL CODE!



"Nygard Stability Patterns" -- Handshaking

"Handshaking is all about letting the server protect itself by throttling its own workload."

-- Chapter 5.6



"Nygard Stability Patterns" -- Handshaking

"Handshaking is all about letting the server protect itself by throttling its own workload."

-- Chapter 5.6

```
function sendOrders(orderList, timeBudget) {  
  if(  
    (health.responseMS+health.latencyMS) < timeBudget  
  ) {  
    bookOrders.send(orderList,timeBudget);  
  }  
  else {  
    error("failed to send orders: HEALTHCHECK");  
  }  
}
```

WARNING: NOT REAL CODE!



"Nygard Stability Patterns" -- Cache

"Caching can reduce the load on the server and cut response times to a fraction of what they would be without caching."

-- Chapter 10.2



"Nygard Stability Patterns" -- Cache

"Caching can reduce the load on the server and cut response times to a fraction of what they would be without caching."

-- Chapter 10.2

```
// server marks data cache-able
function sendResults(response) {
  response.writeHead(status,
    { 'Content-Type' : 'text/plain',
      'Cache-Control': 'public,max-age=108000' }
  );
  response.end(value+'\n');
}
```

WARNING: NOT REAL CODE!



"Nygard Stability Patterns" -- Cache

"Caching can reduce the load on the server and cut response times to a fraction of what they would be without caching."

-- Chapter 10.2

```
// server marks data cache-able
function sendResults(response) {
  response.writeHead(status,
    { 'Content-Type' : 'text/plain',
      'Cache-Control': 'public,max-age=300'
    });
  response.end(value+'\n');
}
```

```
// client manages local cache
function getData(URL) {
  data = null;
  data = cache.read(URL);
  if(!data) {
    data = requestResults(URL);
    cache.write(URL,data);
  }
  return data;
}
```

WARNING: NOT REAL CODE!



Stabilizing Stateless Microservices

Stateless Microservices

```
// http server handling data conversions  
function conversionServer(request, response) {  
    response = convertValue(request);  
    return response;  
}
```

WARNING: NOT REAL CODE!

Networked Stateless

- *What if the work takes too long?*

Stable Stateless Microservices

```
// http server handling data conversions  
function conversionServer(request, response) {  
  if(request.timeBudget > my.averageResponse) {  
    response = FailFastError(request);  
  }  
  else {  
    response = convertValue(request);  
  }  
  return response;  
}
```

1. Fail-Fast

WARNING: NOT REAL CODE!



Stabilizing Persistence Microservices

Persistence Microservices

```
function updateOrders(request, response) {  
  response = localStorage.write(request);  
  return response;  
}
```

WARNING: NOT REAL CODE!

Networked Persistence

- *What if the work takes too long?*
- *What if the dependent service doesn't respond in time?*
- *What if the dependent service is down?*
- *What if the storage overflows (data, logs, etc.)?*

Stable Persistence Microservices

```
function updateOrders(request, response) {  
  if(request.timeBudget < localStorage.latency) {  
    response = FailFastError(request);  
  }  
  else {  
    response = setTimeout(circuitBreaker(  
      localStorage.write(request),  
      {timeout:10,maxFail:3,reset:30}  
    ), timeBudget);  
  }  
  return response;  
}
```

1. Fail-Fast
2. Timeout
3. Circuit Breaker
4. Steady State

WARNING: NOT REAL CODE!



Stabilizing Aggregator Microservices

Aggregator Microservices

```
function writeOrders(request, response) {  
  var resourceList = ["customerDB", "orderDB", "salesDB"]  
  var serviceList = gatherResources(resourceList);  
  response = serviceList(request)  
  
  return response;  
}
```

WARNING: NOT REAL CODE!

Networked Aggregators

- *What if the work takes too long?*
- *What if a dependent services doesn't respond in time?*
- *What if a dependent service is down?*
- *What if storage overflows (data, logs, etc.)?*
- *What if a dependent service is unhealthy?*
- *What if traffic for a service spikes?*

Stable Aggregator Microservices

```
function writeOrders(request, response) {
  var resourceList = ["customerDB", "orderDB", "salesDB"]

  setTimeout(function(request, response, resourceList) {
    var serviceList = gatherResources(resourceList);
    if(serviceList.estimatedCost > request.timeBudget) {
      response = FailFast(request);
    }
    else {
      if(serviceList.healthy === true) {
        circuitBreaker(serviceList, request,
          {timeout:10,maxFail:3,reset:30});
      }
    }
  },request.timeBudget);

  return response;
}
```

1. Fail-Fast
2. Timeout
3. Circuit Breaker
4. Steady State
5. Handshaking
6. Bulkhead

WARNING: NOT REAL CODE!



Nygard's Admonition...

W/ Joe asks:

Is All This Clutter Really Necessary?

You may think, as I did when porting the sockets library, that handling all the possible timeouts creates undue complexity in your code. It certainly adds complexity. You may find that half your code is devoted to error handling instead of providing features. I argue, however, that the essence of aiming for production—instead of aiming for QA—is handling the slings and arrows of outrageous fortune. That error-handling code, if done well, adds resilience. Your users may not thank you for it, because nobody notices when a system *doesn't* go down, but you will sleep better at night.

Applying Nygard's Patterns to Services

- **Stateless**
 - *fail fast*
- **Persistence**
 - *fail fast, timeout, circuit breaker, steady state*
- **Aggregation**
 - *fail fast, timeout, circuit breaker, steady state, handshaking, bulkhead*

**Apply Nygard's Stability Patterns
to improve the health
of your components and your system.**



A stylized illustration of a television set. The screen is filled with a large, jagged yellow starburst shape. Inside the starburst, the text "BUT WAIT, there's more!" is written in a bold, red, sans-serif font, slanted upwards. The television's frame is a solid grey color. On the right side of the frame, there are two large circular speaker grilles stacked vertically, two smaller circular buttons below them, and a rectangular control panel at the bottom right.

**BUT WAIT,
there's more!**

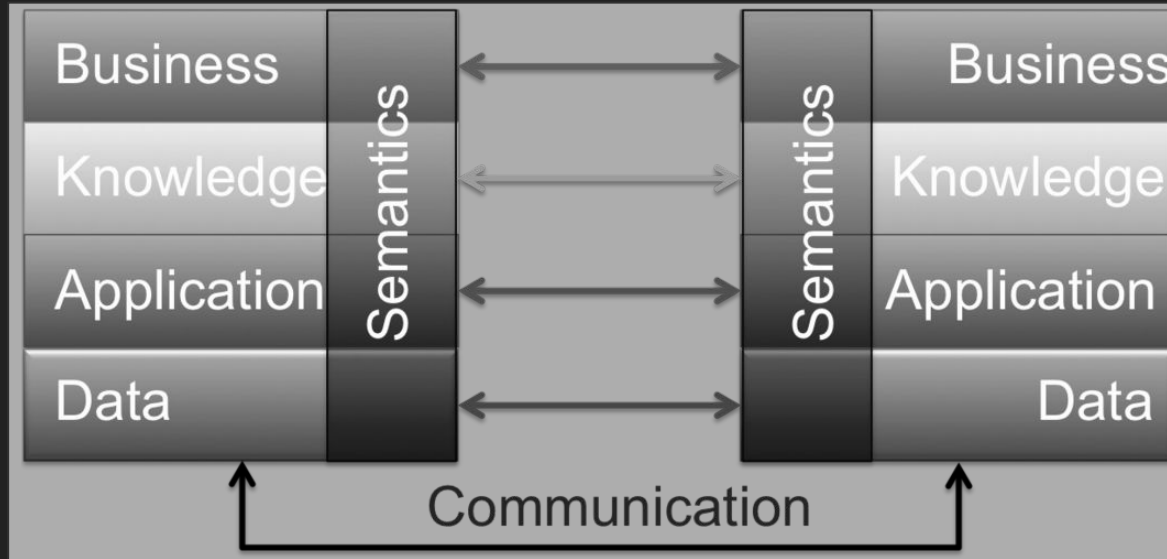
Aim for Interop, not Integration...

"Interoperation is peer to peer. Integration is where a system is subsumed within another."

-- Michael Platt, Microsoft



Aim for Interop, not Integration...



Include time/distance in your models

"There is no simultaneity at a distance."

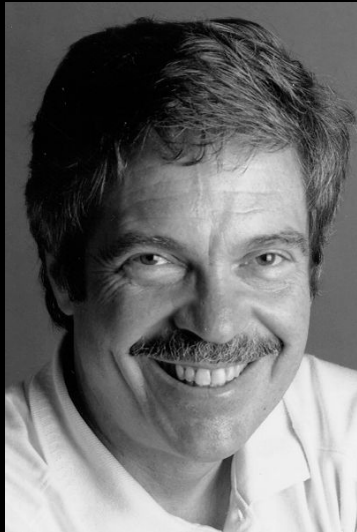
-- Pat Helland, Salesforce



Pat Helland

Include time/distance in your models

"I'm sorry that coined the term 'objects' for this topic. The big idea is 'messaging'."



Alan Kay, 1998

Remember, you're programming the *network*

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

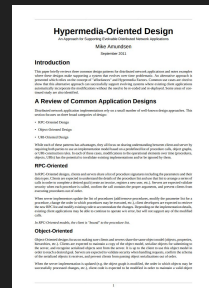


L Peter Deutsch

Remember, you're programming the *network*

- **Safety**

<https://www.w3.org/2011/10/integration-workshop/p/hypermedia-oriented-design.pdf>



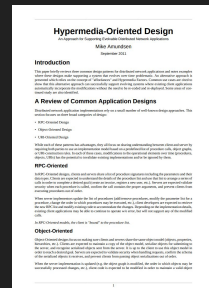
Remember, you're programming the *network*

● Safety

The HTTP protocol supports a number of "safe" actions such as HEAD, and GET.

The HTTP methods PUT, POST, and DELETE are categorized as "unsafe" actions.

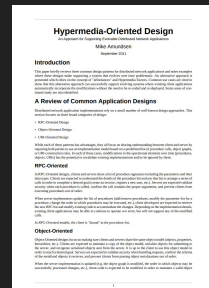
<https://www.w3.org/2011/10/integration-workshop/p/hypermedia-oriented-design.pdf>



Remember, you're programming the *network*

- Safety
- Idempotence

<https://www.w3.org/2011/10/integration-workshop/p/hypermedia-oriented-design.pdf>



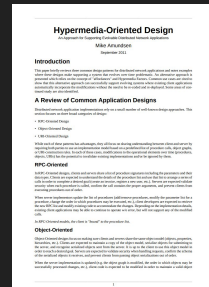
Remember, you're programming the *network*

- Safety
- Idempotence

In HTML when a FORM element has the METHOD property set to "get" this represents an idempotent action.

When the same property is set to "post" the affordance represents a non-idempotent action.

<https://www.w3.org/2011/10/integration-workshop/p/hypermedia-oriented-design.pdf>

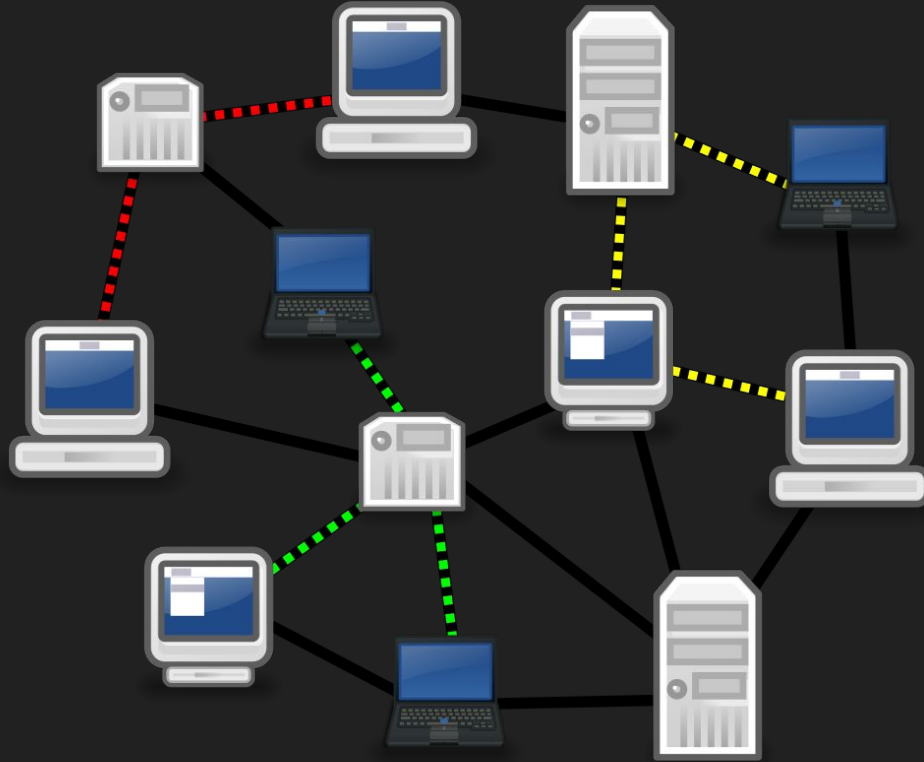


Other Considerations...

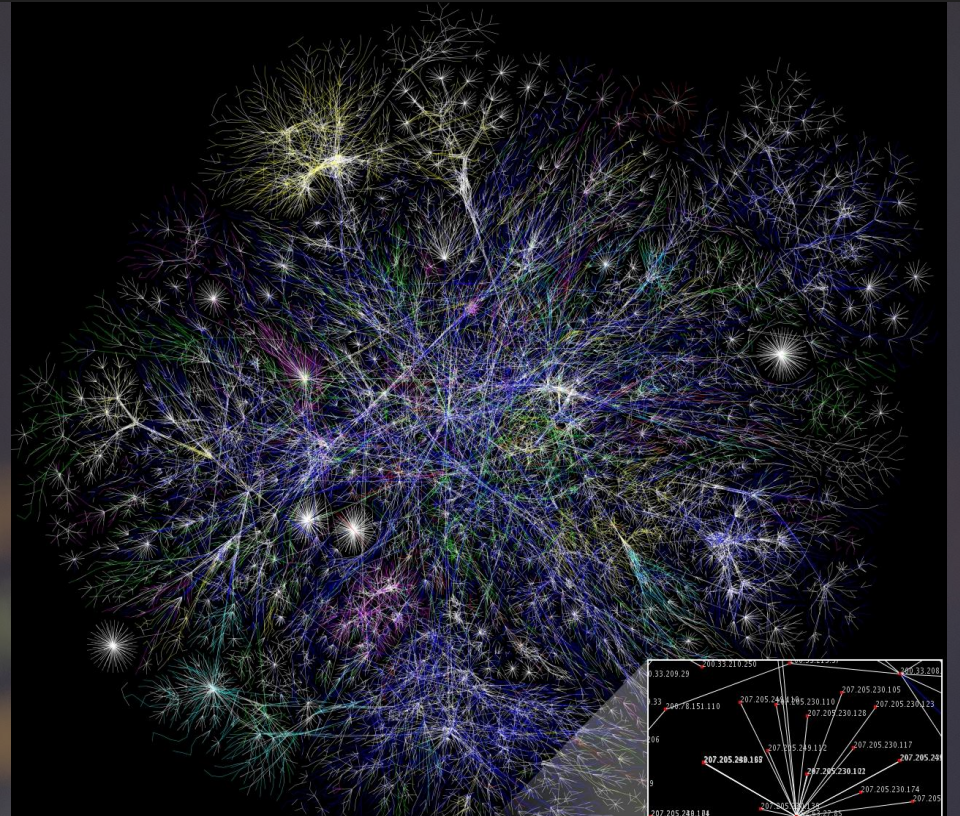
- Interop vs. Integration
- Time & Distance
- Safety & Idempotence

So...

We need microservices...



So that we can program the network...



Which means applying patterns to our interactions..,

```
function writeOrders(request, response) {
  var resourceList = ["customerDB", "orderDB", "salesDB"]

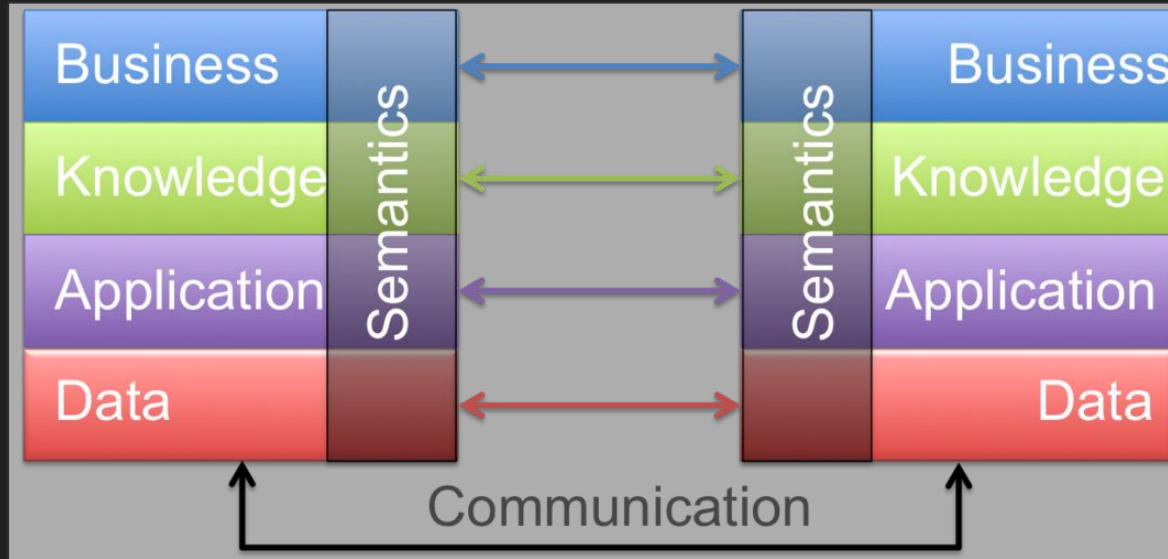
  setTimeout(function(request, response, resourceList) {
    var serviceList = gatherResources(resourceList);
    if(serviceList.estimatedCost > request.timeBudget) {
      response = FailFast(request);
    }
    else {
      if(serviceList.healthy === true) {
        circuitBreaker(serviceList, request,
          {timeout:10,maxFail:3,reset:30});
      }
    }
  },request.timeBudget);

  return response;
}
```

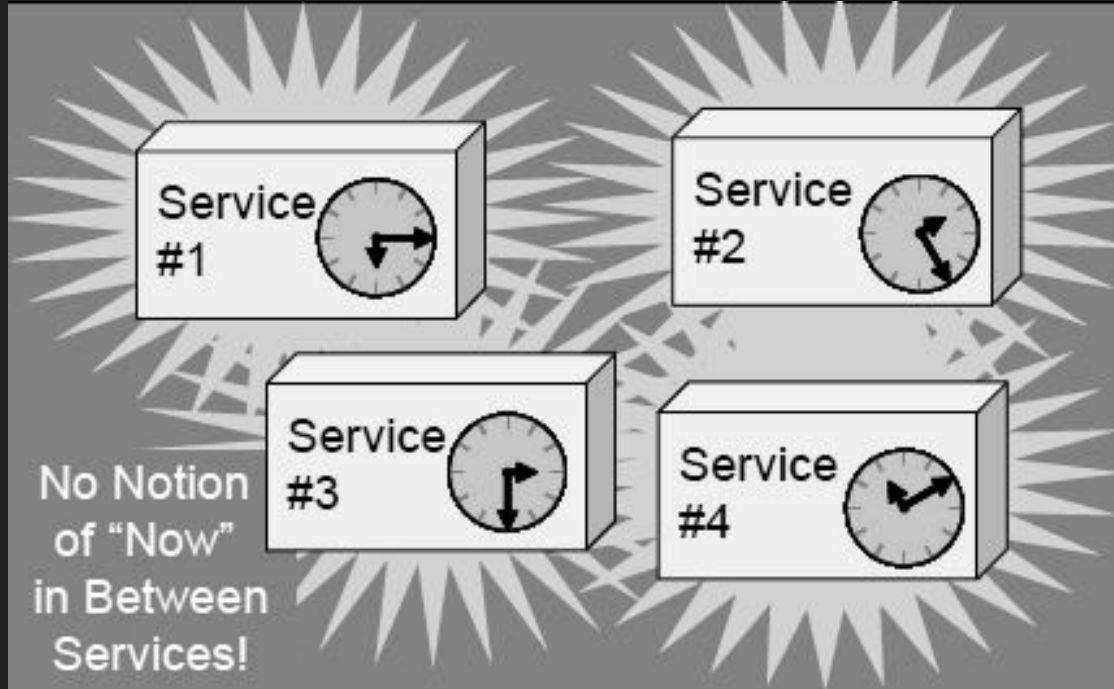
1. Fail-Fast
2. Timeout
3. Circuit Breaker
4. Steady State
5. Handshaking
6. Bulkhead



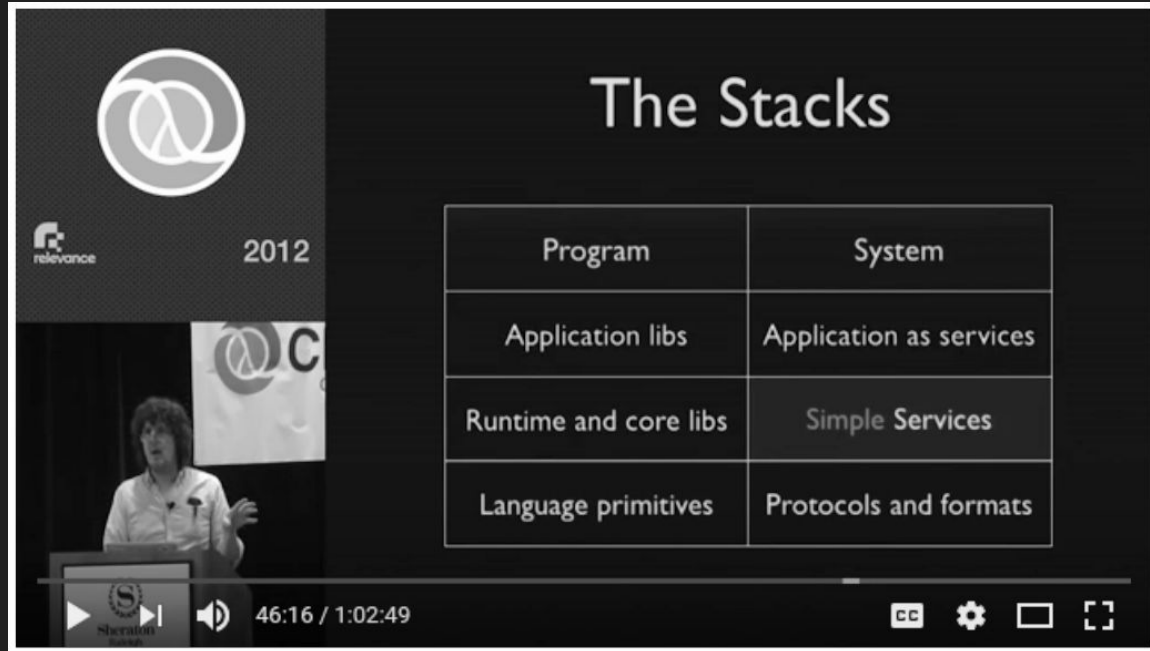
And that means understanding the role of semantics...



And the role of distance & time...



And constantly reminding ourselves of the challenge.



The screenshot shows a video player interface. In the top left corner, there is a logo consisting of two overlapping circles, the 'relevance' logo, and the year '2012'. Below this, a smaller version of the same logo is visible on a screen behind a speaker. The speaker is a woman with short dark hair, wearing a light-colored shirt, standing at a podium with a microphone. The main content of the video is a slide titled 'The Stacks' which contains a table with two columns: 'Program' and 'System'. The table has four rows of content. At the bottom of the video player, there is a control bar with a play button, a volume icon, a progress bar showing '46:16 / 1:02:49', and icons for closed captions, settings, and full screen.

Program	System
Application libs	Application as services
Runtime and core libs	Simple Services
Language primitives	Protocols and formats

That's a lot!

The Best Software Architecture

"The best software architecture 'knows' what changes often and makes that easy."

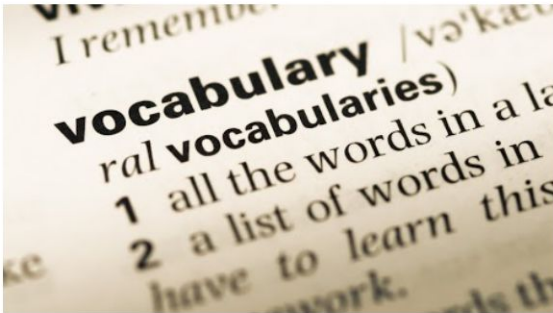
- Paul Clements



Thank you.

<http://g.mamund.com/at-newsletter-signup>

Subscribe



[Newsletter Sign-Up Page](#)

Free Amundsen Talks Newsletter

Stay Up-to-Date

Keep up with the latest Microservice and API news with the **Amundsen Talks (AT) Newsletter**. Where is Mike Amundsen this month? What's the latest news from the field? Stay up-to-date with the AT.

Original Content

Every month, you'll get a collection of links to recent articles, updates on Mike's travels to conferences and shows around the world, and at least one original article or paper.

Bonus Issue

Don't miss an issue. Sign up now and get a special "Welcome to the AT" bonus issue right away.

Learning the Three Types of Microservices

Please fill out the session survey in the CodeMash app!

CodeMash 2019

Mike Amundsen

@mamund

API Academy