Using Erlang in a Carrier-Grade Media Distribution Switch

Steve Vinoski
Member of Technical Staff
Verivue, Inc.
Westford, MA USA

Erlang Factory Sf Bay Area 2010
25 March 2010
http://steve.vinoski.net/

Agenda

- Media distribution market
- Verivue's systems
- Where we use Erlang
- Advice for new Erlang users

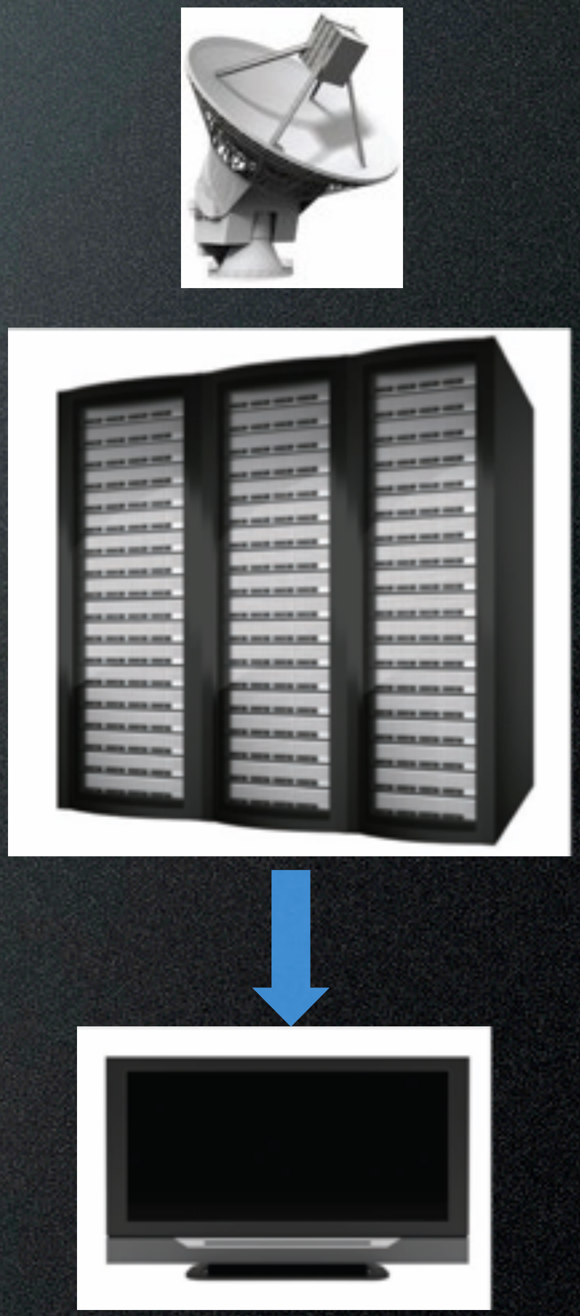
How I Started With Erlang

- First 6 years of my career: hardware test engineer
- Next 17 years: enterprise middleware
- Left middleware in 2007 to join Verivue, a media distribution startup
- Part of the reason I made the change: so I could use Erlang

Video Delivery Trends

Video Delivery Trends

Managed Video Network

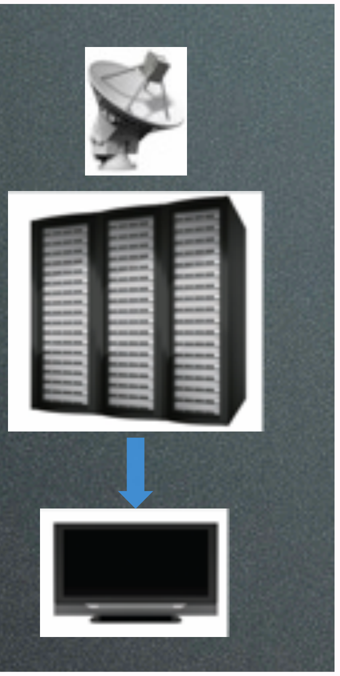


- Content replicated locally
- “Push” model
- Limited choice
- High quality experience
- Limited targeting

Video Delivery Trends

Managed Video Network

- Content replicated locally
- "Push" model
- Limited choice
- High quality experience
- Limited targeting

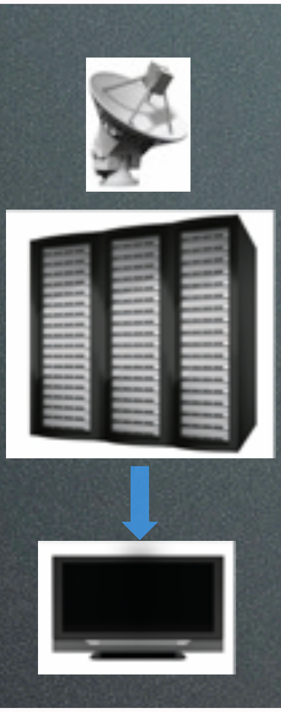


abc viacom

Video Delivery Trends

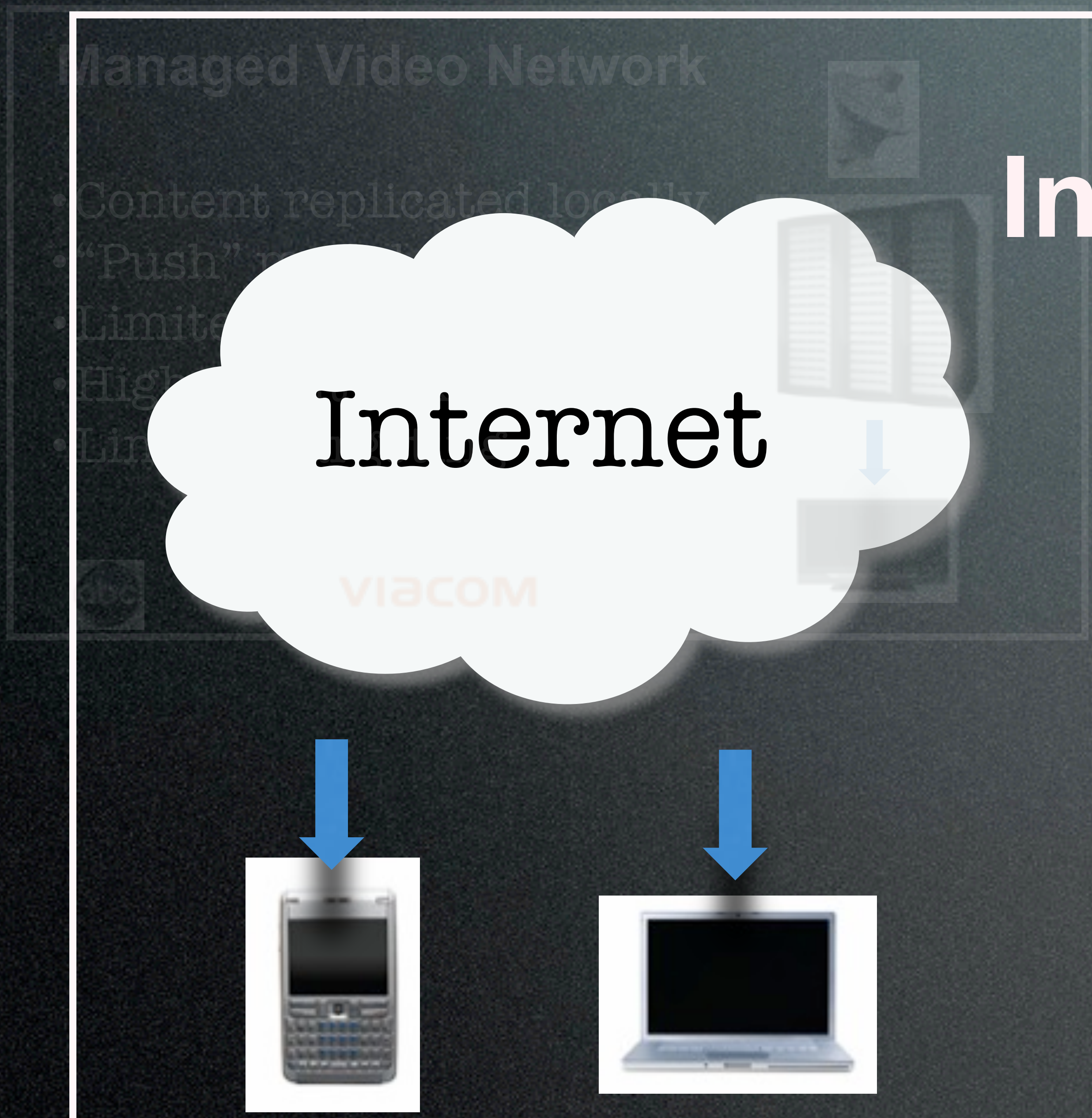
Managed Video Network

- Content replicated locally
- “Push” model
- Limited choice
- High quality experience
- Limited targeting



abc VIACOM

Video Delivery Trends



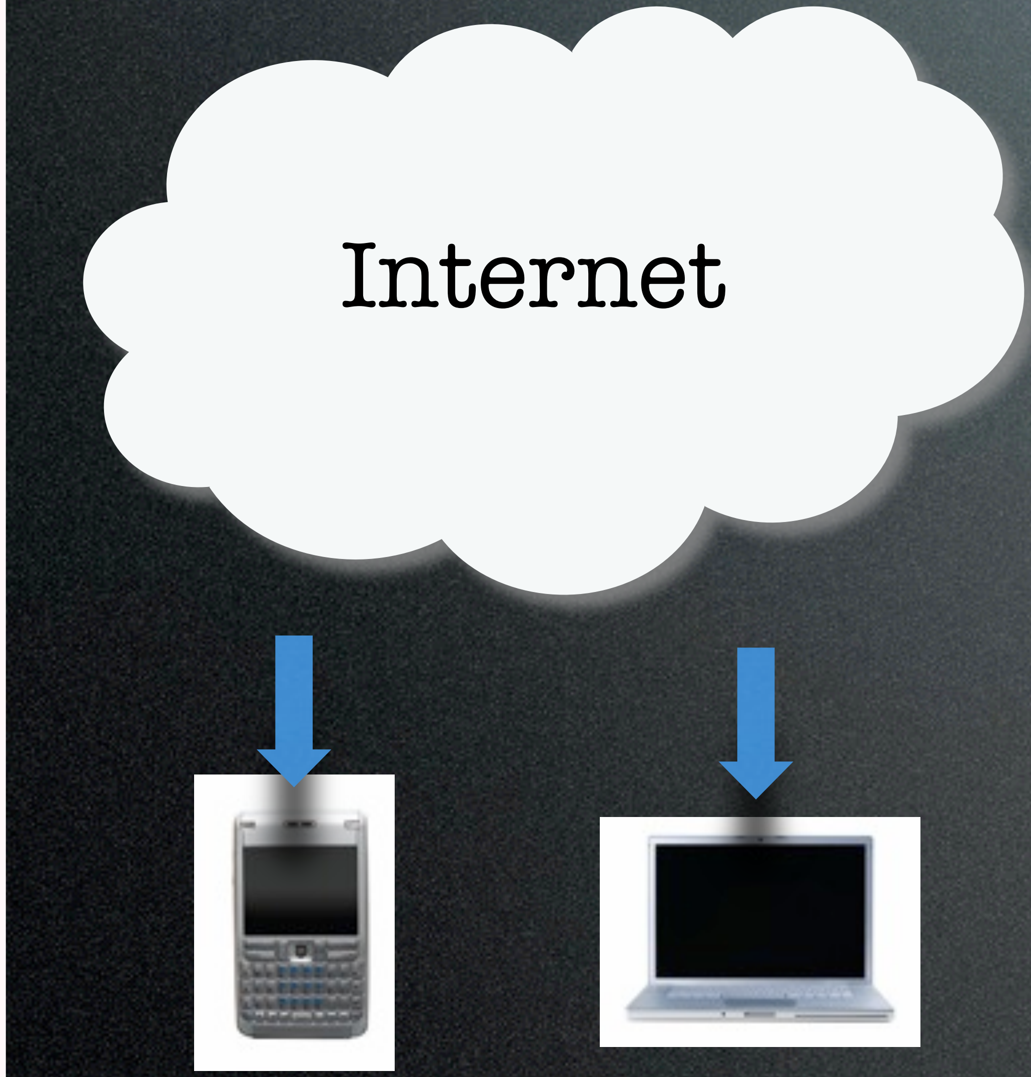
Internet Video

- "Pull" model
- Unlimited choice
- Highly targeted
- PC and Mobile

YouTube Broadcast Yourself™ hulu™

Video Delivery Trends

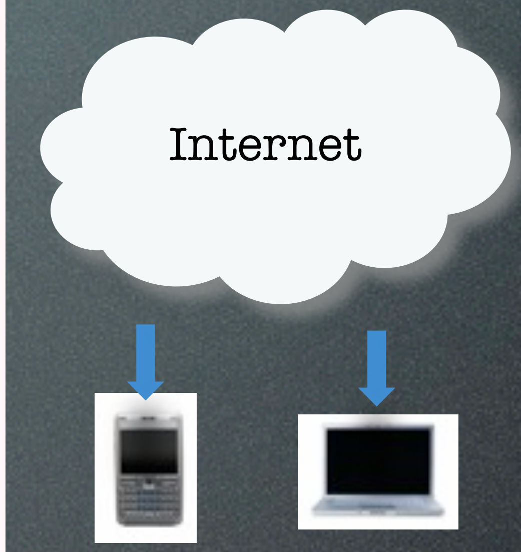
Internet Video



- "Pull" model
- Unlimited choice
- Highly targeted
- PC and Mobile

YouTube Broadcast Yourself™ hulu™

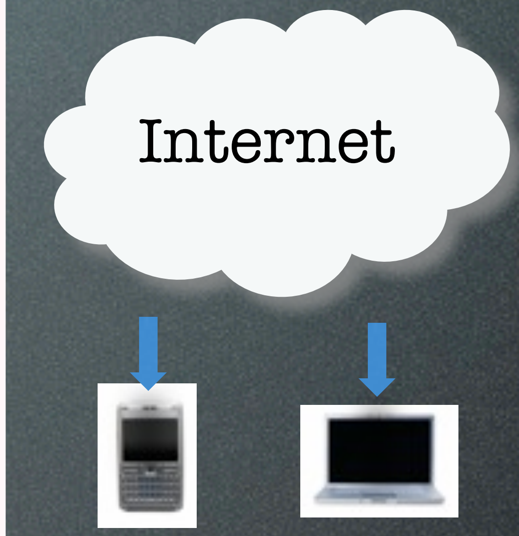
Video Delivery Trends



Internet Video

- "Pull" model
- Unlimited choice
- Highly targeted
- PC and Mobile

Video Delivery Trends



Internet Video

- "Pull" model
- Unlimited choice
- Highly targeted
- PC and Mobile

Video Delivery Trends

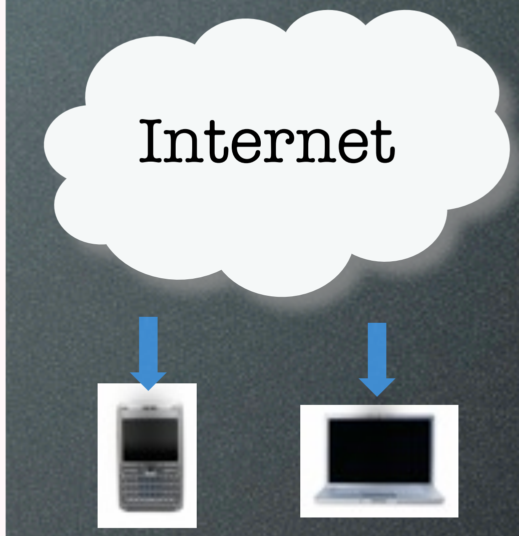
Managed Video Network

- Content replicated locally
- "Push" model
- Limited choice
- High quality experience
- Limited targeting

abc

viacom

Internet Video



Internet

- "Pull" model
- Unlimited choice
- Highly targeted
- PC and Mobile

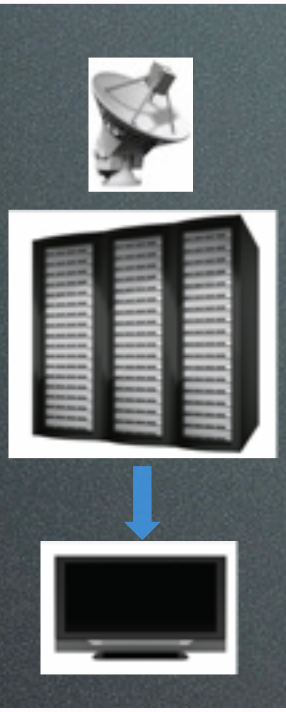
YouTube Broadcast Yourself™

hulu™

Video Delivery Trends

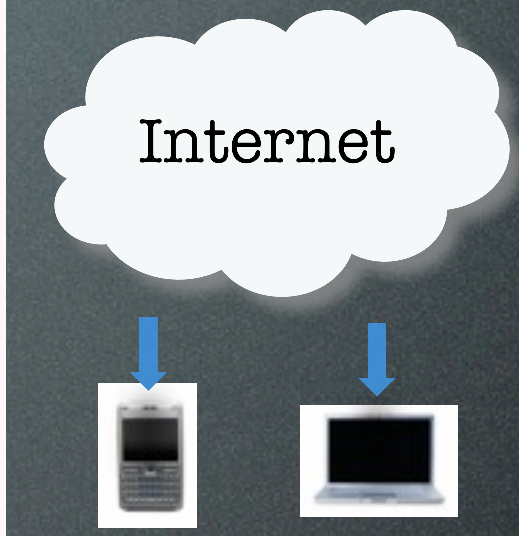
Managed Video Network

- Content replicated locally
- “Push” model
- Limited choice
- High quality experience
- Limited targeting



abc VIACOM

Internet Video

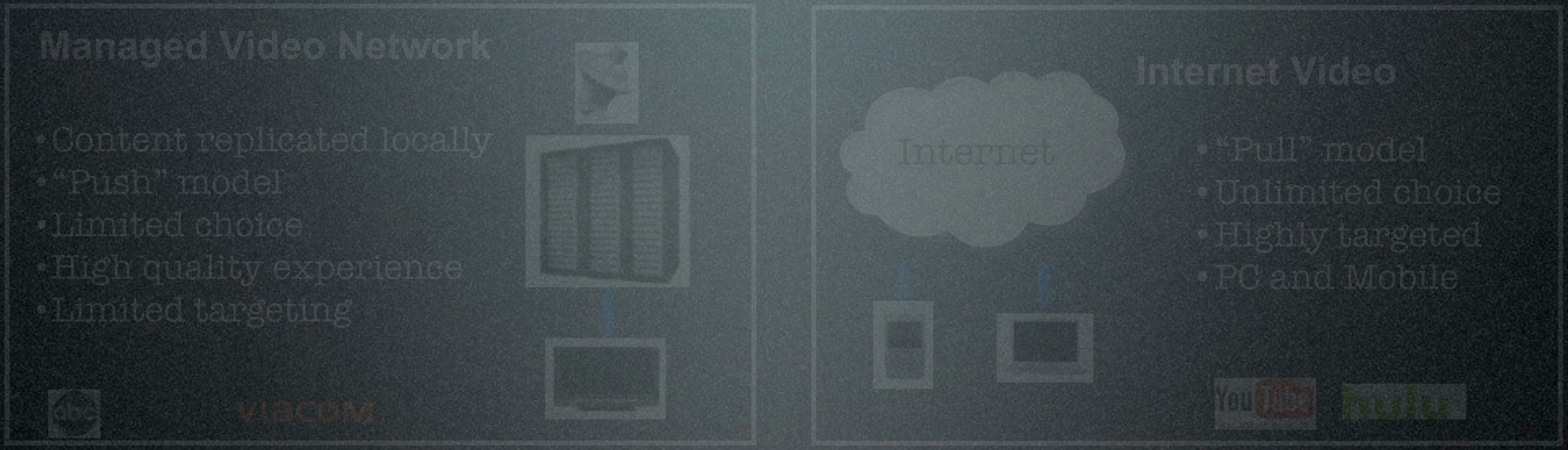


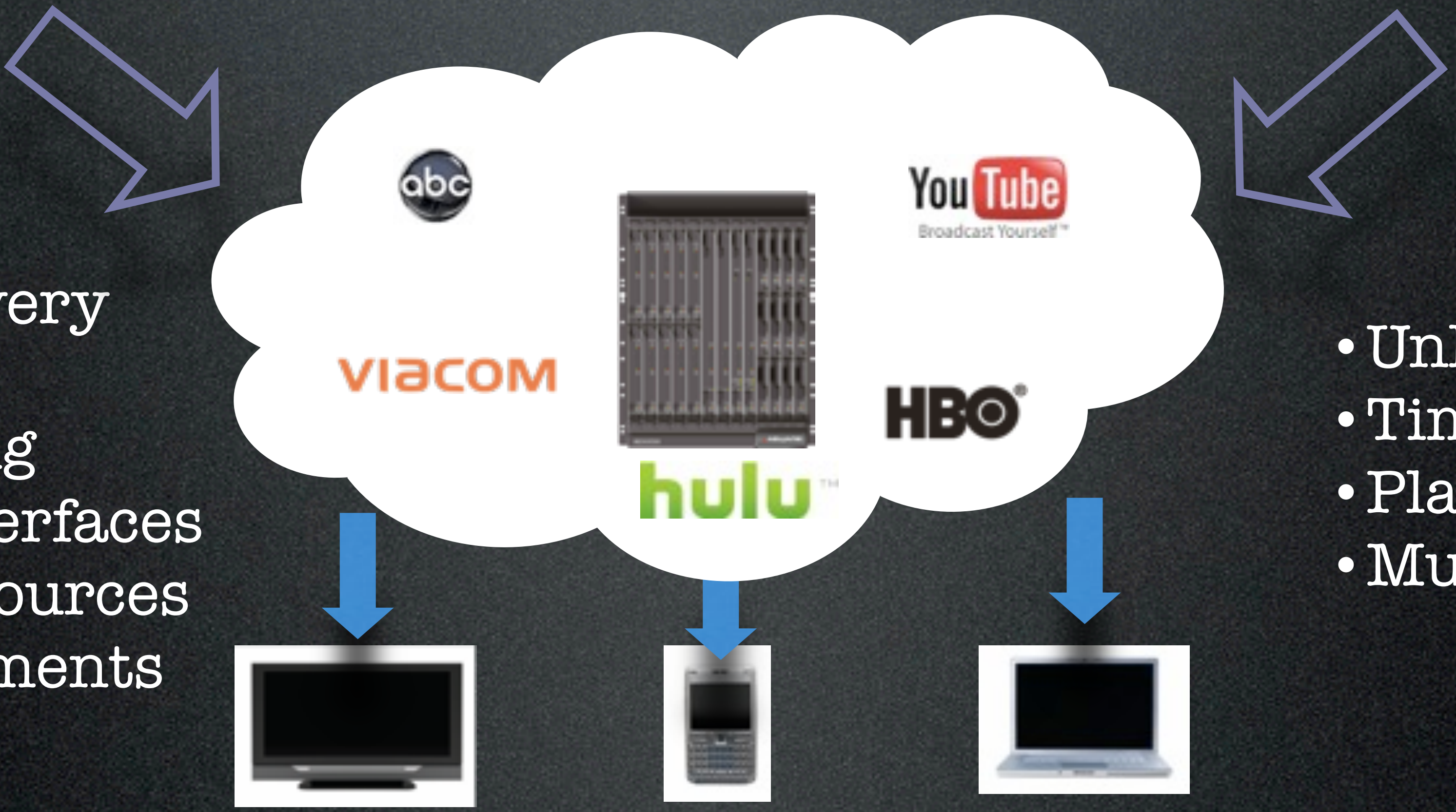
Internet

- “Pull” model
- Unlimited choice
- Highly targeted
- PC and Mobile

YouTube hulu

Video Delivery Trends





- Open content delivery network
- Distributed caching
- Open standard interfaces
- Multiple content sources
- Multi-protocol elements

- Unlimited content
- Time-shifting
- Place-shifting
- Multiple screens

Video Delivery Problems

- PC servers and disk-based storage systems not well suited for growing surge of video traffic
- Off-the-shelf enterprise-class servers not optimized for content delivery
- Hard disk drives are not cost effective for high-bandwidth content delivery
- Servers need to be augmented by networking gear such as firewalls, switches and load balancers

Video Delivery Problems

- PC servers and disk-based storage systems not well suited for growing surge of video traffic
- Off-the-shelf enterprise-class servers not optimized for content delivery
- Hard disk drives are not cost effective for high-bandwidth content delivery
- Servers need to be augmented by networking gear such as firewalls, switches and load balancers

Video Delivery Problems

Video Delivery Problems

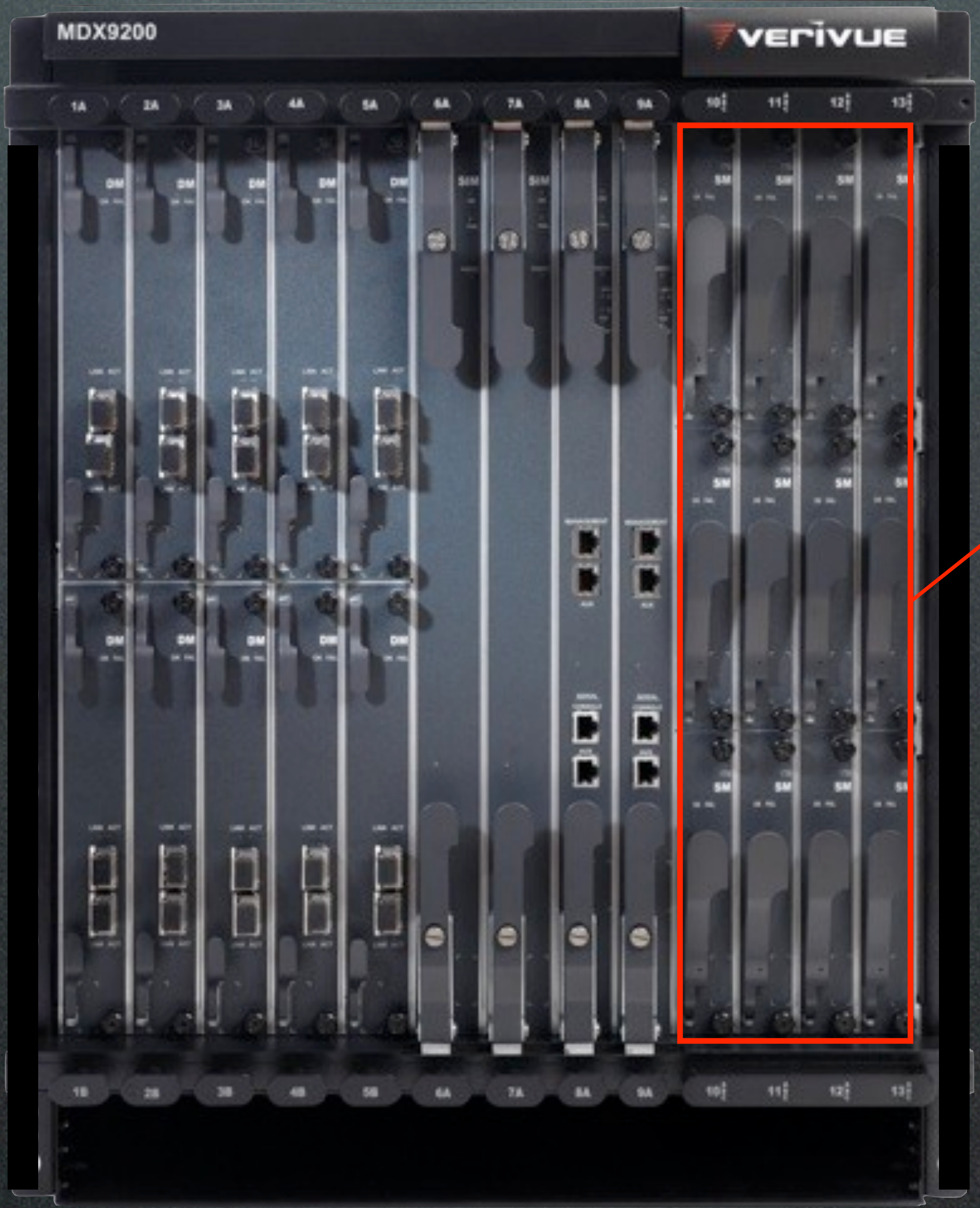
Video Delivery Problems

Meeting increasing demand for content delivery with traditional server elements presents financial and operational challenges

Verivue MDX 9000 Series

- Media Distribution Switch
- High-capacity content delivery
- Large solid-state local storage
- Integrated networking capabilities
- Lower operational overhead and power requirements
- Eliminate datacenter sprawl

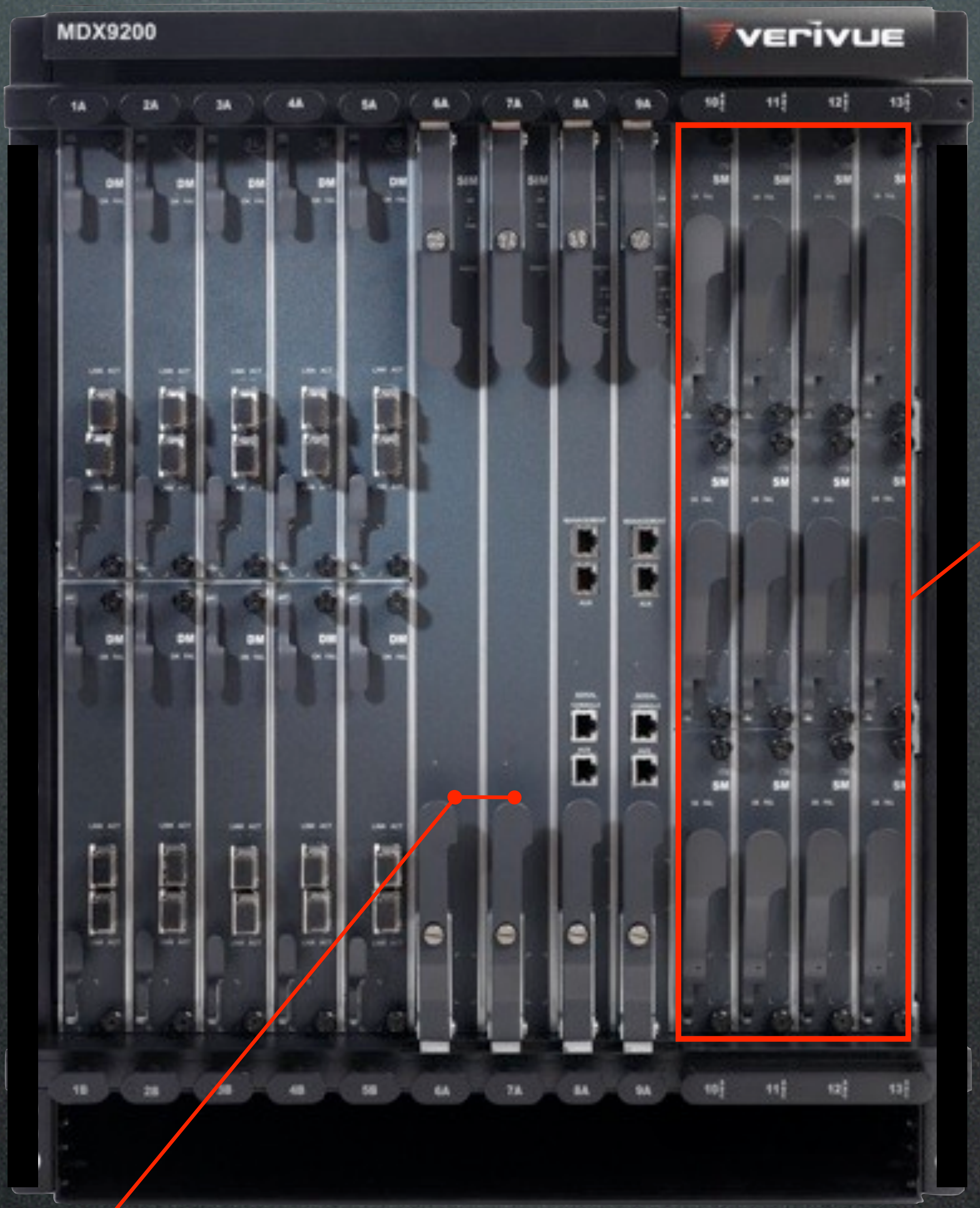
Verivue MDX 9200



Storage Modules (SM)
Up to 12 flash memory blades available in 2 or 4 TB

2-48 TB storage capacity

Verivue MDX 9200



Storage Modules (SM)
Up to 12 flash memory blades available in 2 or 4 TB

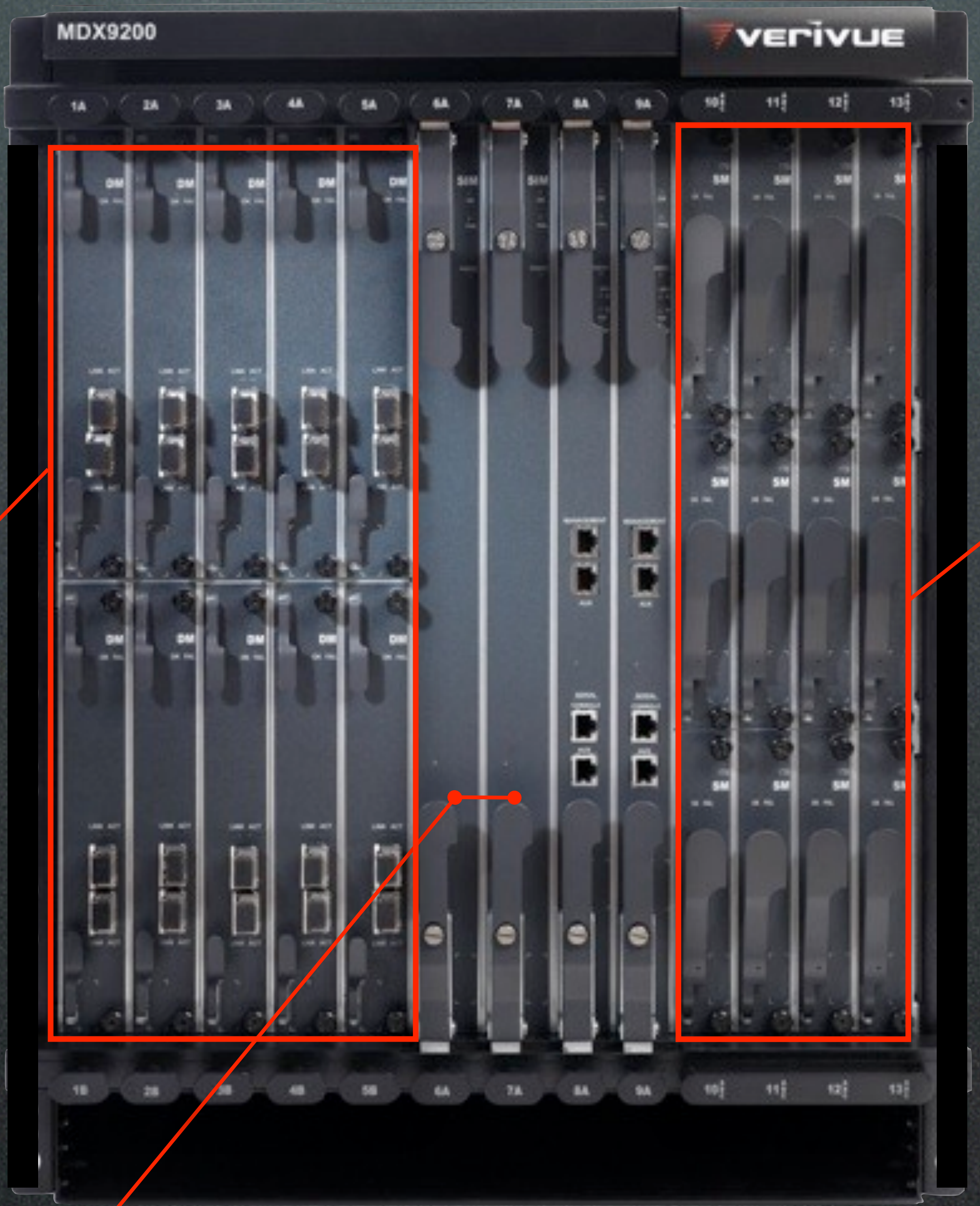
2-48 TB storage capacity

Switch/Ingest Module (SIM)
9.6 Gb/s continuous write bandwidth

Verivue MDX 9200

20-200 Gb/s delivery capacity

Storage Modules (SM)
Up to 12 flash memory blades available in 2 or 4 TB

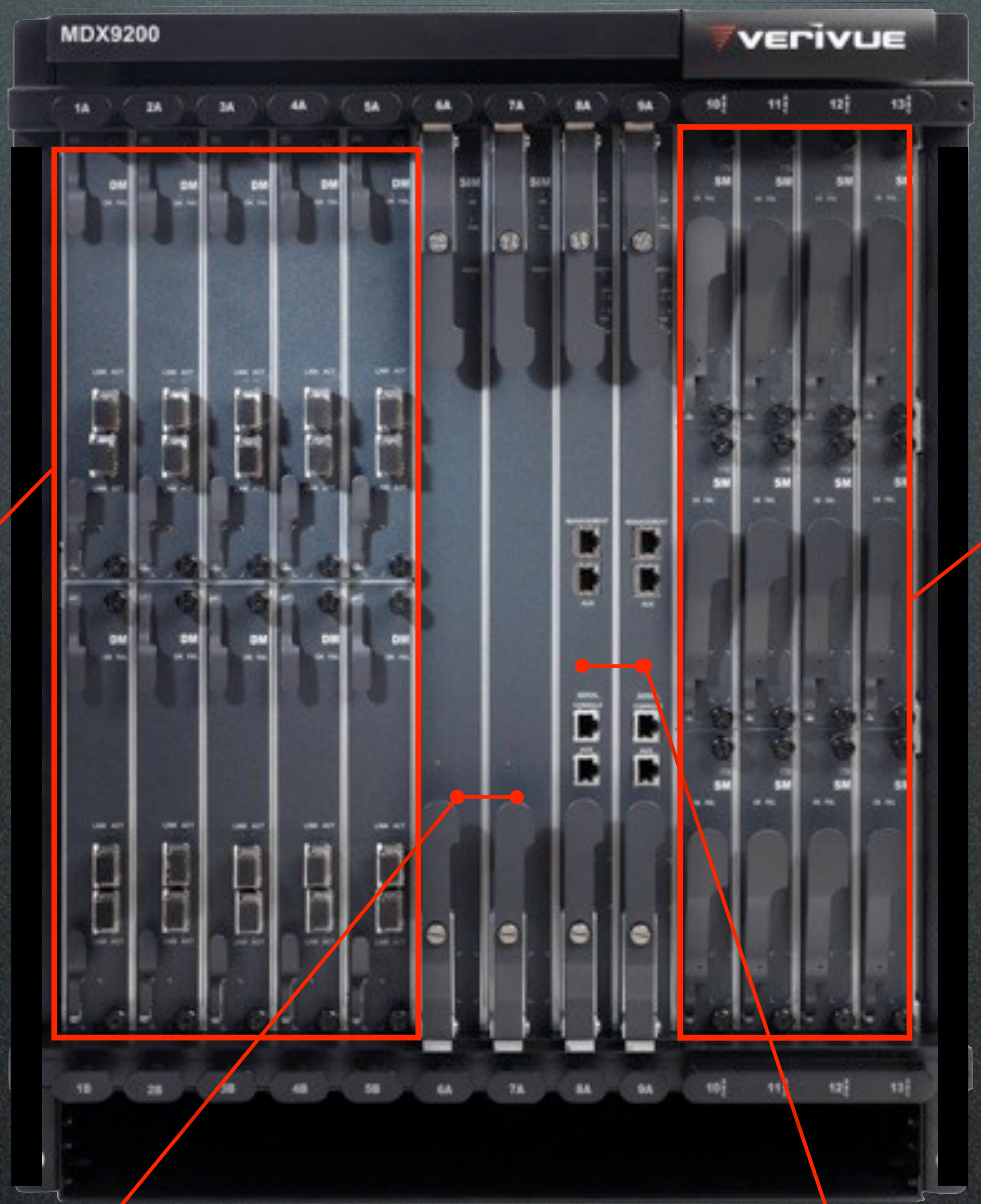


Delivery Modules (DM)
Up to 10 blades, each providing up to 20 Gb/s of broadband delivery

2-48 TB storage capacity

Switch/Ingest Module (SIM)
9.6 Gb/s continuous write bandwidth

Verivue MDX 9200



20-200 Gb/s delivery capacity

Delivery Modules (DM)
Up to 10 blades, each providing up to 20 Gb/s of broadband delivery

Switch/Ingest Module (SIM)
9.6 Gb/s continuous write bandwidth

Storage Modules (SM)
Up to 12 flash memory blades available in 2 or 4 TB

2-48 TB storage capacity

Switch/Control Module (SCM)
200 Gb/s switch fabric

Verivue MDX 9000 Series

MDX 9200

20-200 Gb/s
2-48 TB

MDX 9020

10-20 Gb/s
4-8 TB

System Characteristics

- Embedded Linux
- Multiple cards, with redundancy
- Intra-chassis network (ICN)
  - software components communicate over ICN
  - many components written in C++

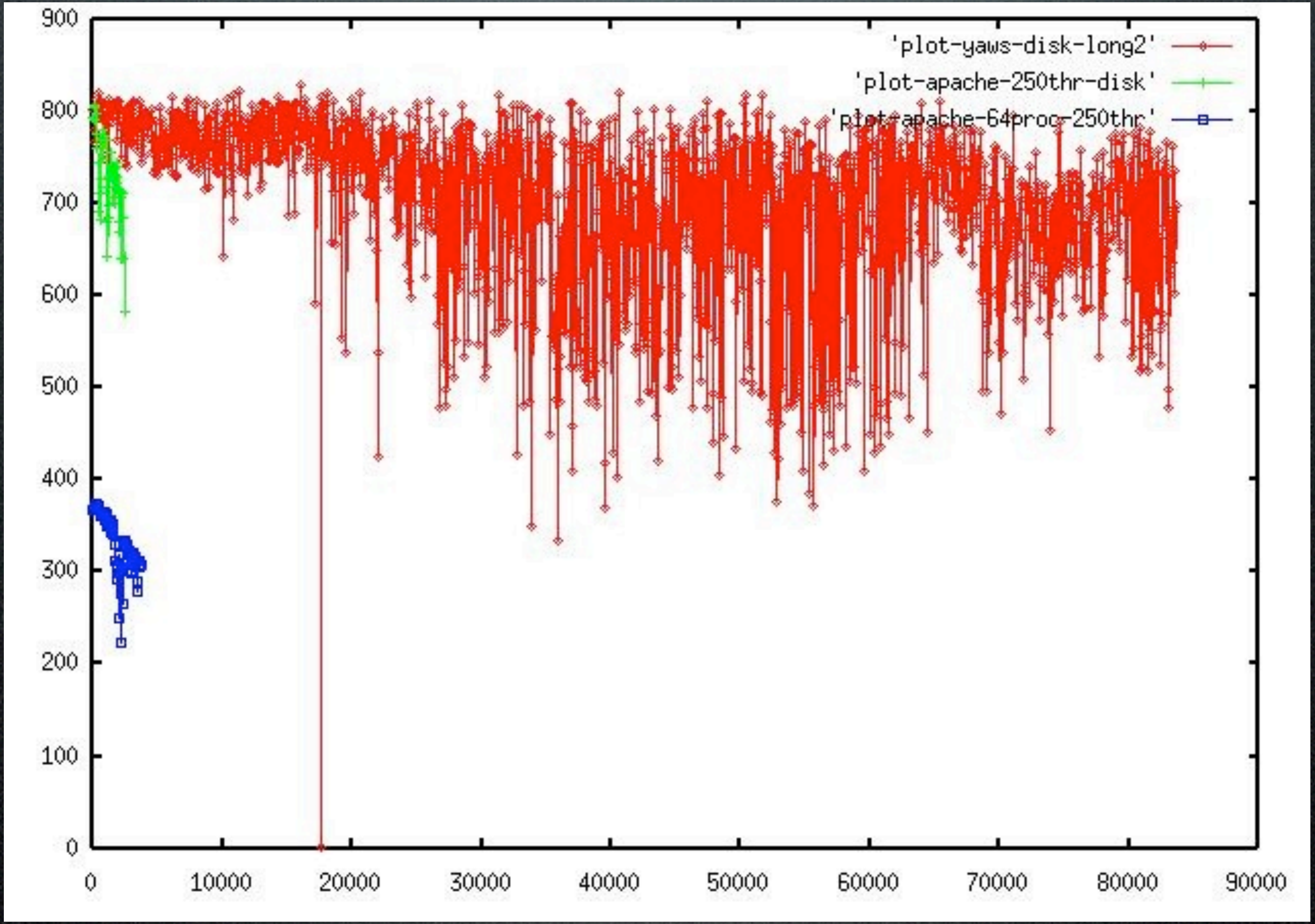
Where We Use Erlang

- HTTP-based content ingest and delivery (control plane)
- HTTP cache functionality (control plane)
- System provisioning and configuration
- Integration w/ 3rd-party asset management interfaces (typically XML or JSON over HTTP)
- Content index parsing and management

Initial Erlang Work

- Initial plans included set top box (STB) support
    - HTTP-based support for electronic program guide (EPG) and video-on-demand (VOD) purchase
    - tens of thousands of STB connections

Apache vs. Yaws



Contributions to Yaws

- I started using Yaws, submitted some patches, and Klacke adopted me :-)
- Wrote a Yaws sendfile driver, to reduce CPU usage during file delivery
- Wrote Yaws support for long-polling clients (for COMET apps)
- Fix the occasional bug as well

Cross-Stream Bookmarking

Scalable Media Distribution
Seamless Multi-Screen Convergence

- Start viewing a show on TV...
- ...then shift viewing over to laptop
- ...or shift viewing to phone

Cross-Stream Bookmarks with Erlang

- In the early days, an important customer wanted a “three screen” demo
- At the time we had only implemented traditional VOD delivery (to the TV screen)
- Had only 4 days to get the other two screens working

Requirements

Requirements

- Content delivered to laptop and phone must pick up where VOD delivery leaves off

Requirements

- Content delivered to laptop and phone must pick up where VOD delivery leaves off
- Web delivery must account for media framing when delivering desired content range

Requirements

- Content delivered to laptop and phone must pick up where VOD delivery leaves off
- Web delivery must account for media framing when delivering desired content range
- Also need to pace content delivery for some mobile devices

How Erlang Helped

How Erlang Helped

- Bit syntax: decode video content to find frame at the desired time offset

How Erlang Helped

- Bit syntax: decode video content to find frame at the desired time offset
- Pacing content delivery easy with Yaws streaming API

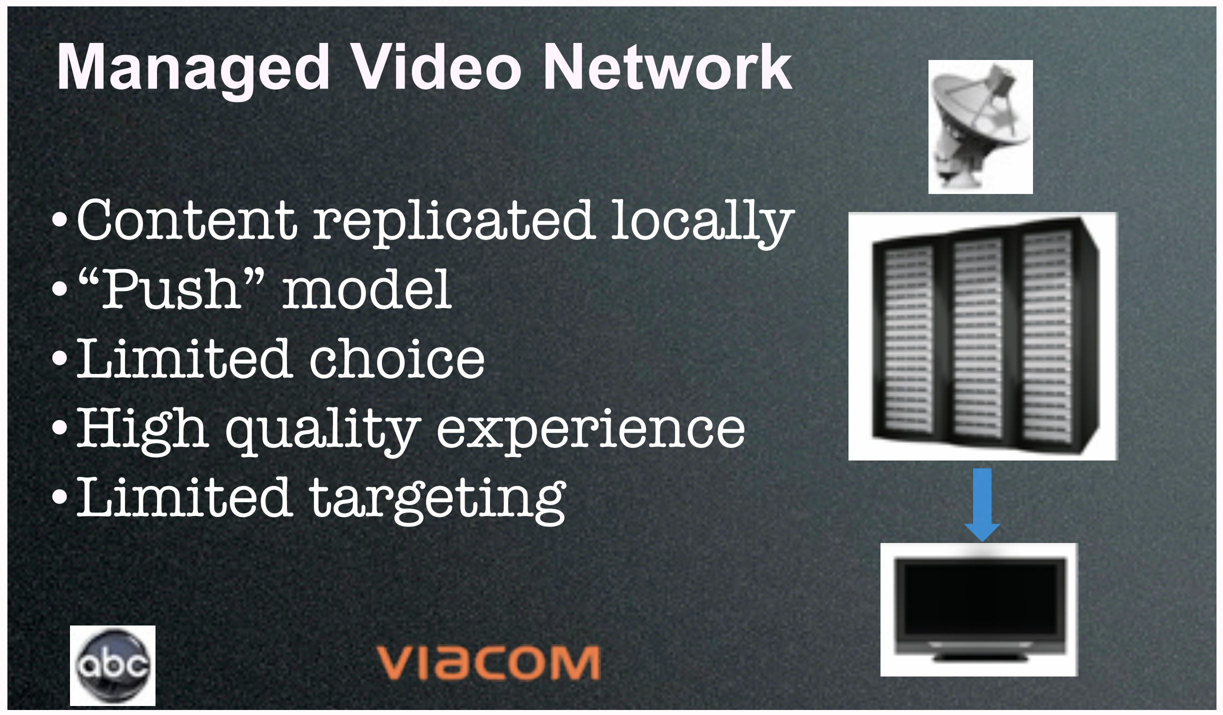
How Erlang Helped

- Bit syntax: decode video content to find frame at the desired time offset
- Pacing content delivery easy with Yaws streaming API
- Yaws makes it easy to handle different media types for different HTTP clients

How Erlang Helped

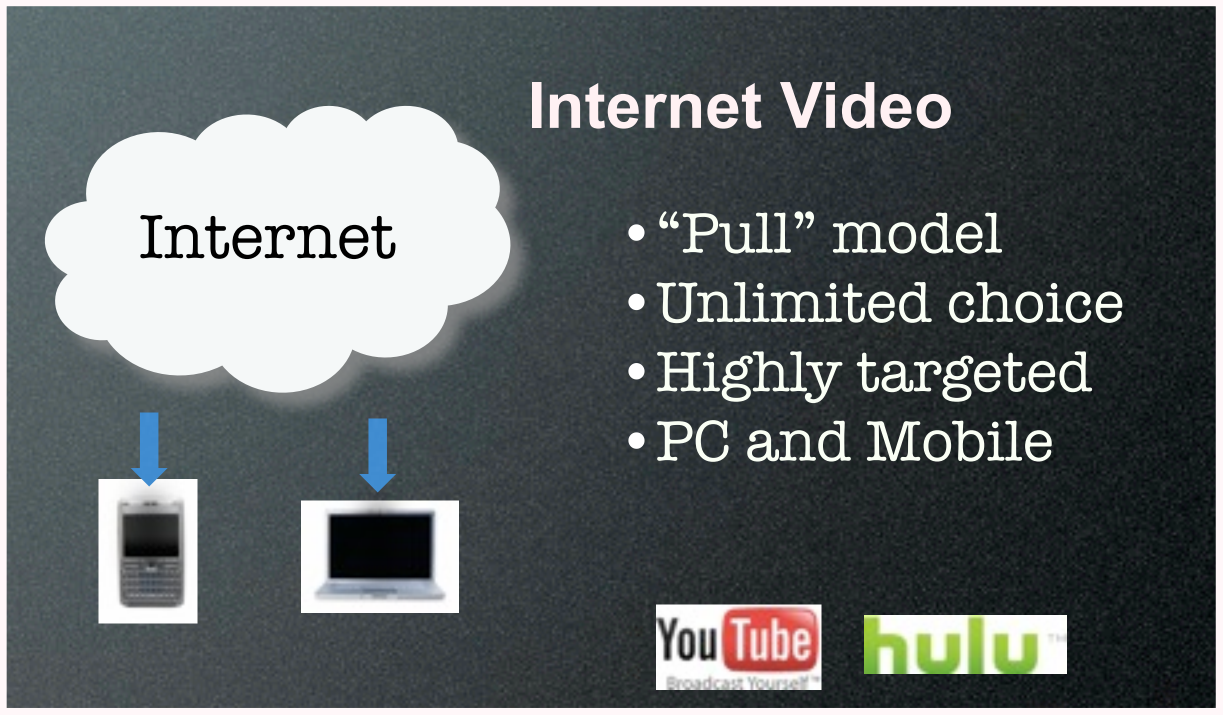
- Bit syntax: decode video content to find frame at the desired time offset
- Pacing content delivery easy with Yaws streaming API
- Yaws makes it easy to handle different media types for different HTTP clients
- Yes, we made the 4-day deadline :-)

HTTP Capabilities



Managed Video Network

- Content replicated locally
- “Push” model
- Limited choice
- High quality experience
- Limited targeting



Internet Video

- “Pull” model
- Unlimited choice
- Highly targeted
- PC and Mobile

- HTTP required for both areas
  - control and management for VOD
  - actual delivery for the Web case

HTTP and VOD

- Used for management, not for delivery
- E.g., we jointly developed HTTP content management with another vendor
    - allows cable operators to ingest content, list what’s stored, delete
    - quickly implemented with Yaws and xmerl

Thinking about Integration

- The need to integrate is a given
- Newer stuff tends to be HTTP-based
- Older stuff like SOAP and CORBA still around (e.g. Time-Warner ISA)
- Reminds me of old middleware days

Integration Using Erlang

Integration Using Erlang

- Integration often involves distribution

Integration Using Erlang

- Integration often involves distribution
- Dealing with data: bit syntax, built-in packet decoders (HTTP, FCGI, CDR)

Integration Using Erlang

- Integration often involves distribution
- Dealing with data: bit syntax, built-in packet decoders (HTTP, FCGI, CDR)
- Trivial access to TCP, UDP

Integration Using Erlang

- Integration often involves distribution
- Dealing with data: bit syntax, built-in packet decoders (HTTP, FCGI, CDR)
- Trivial access to TCP, UDP
- Sync or async, easy event handling

Integration Using Erlang

- Integration often involves distribution
- Dealing with data: bit syntax, built-in packet decoders (HTTP, FCGI, CDR)
- Trivial access to TCP, UDP
- Sync or async, easy event handling
- Application protocol handlers built using gen_server or gen_fsm

Integration Using Erlang

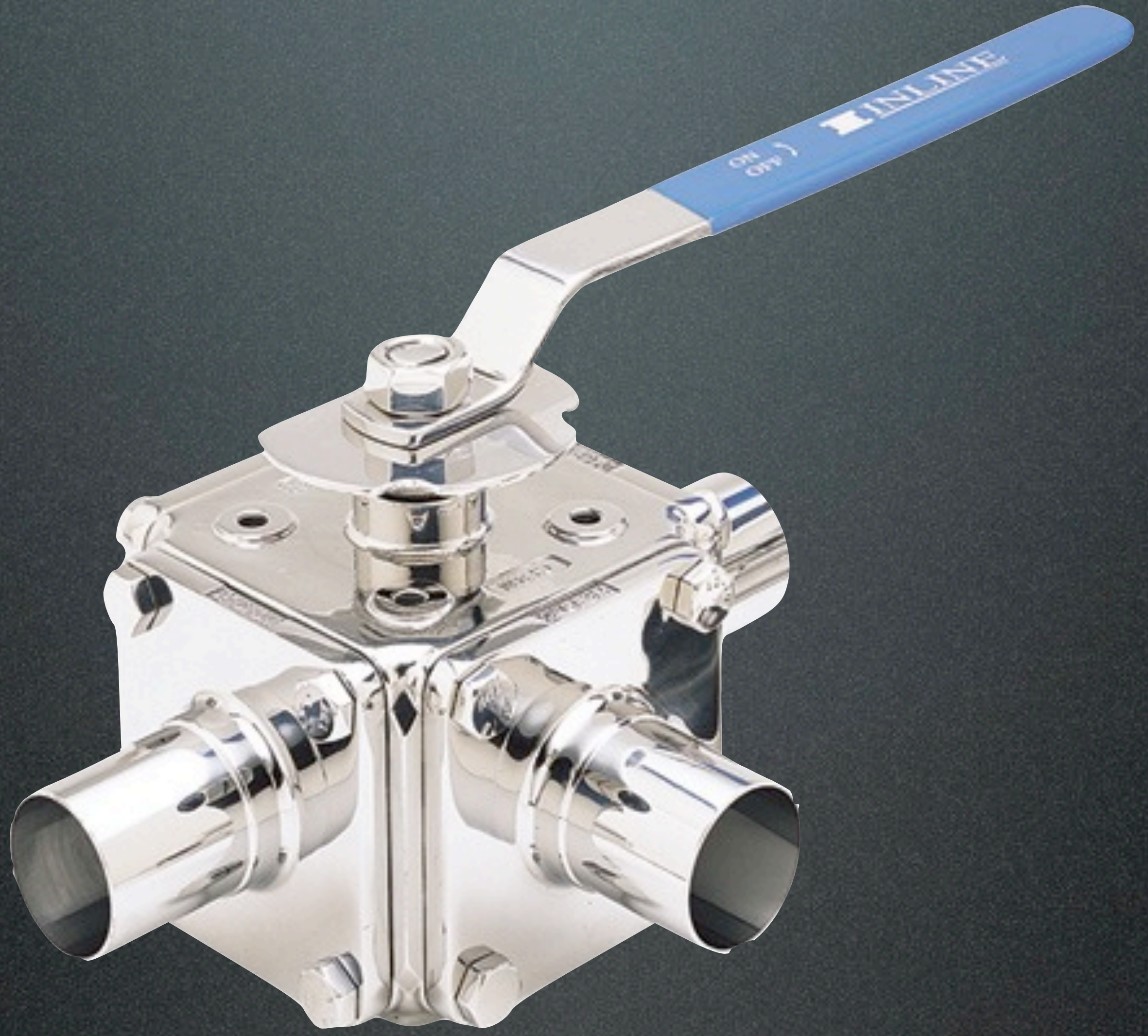
Integration Using Erlang

- Often write little networked clients and servers directly in the erl shell
- Packet decoding and bit syntax sets Erlang apart from netcat, perl, etc. in this regard
- It's like a middleware/coordination DSL
- Critical for testing (more on this later)

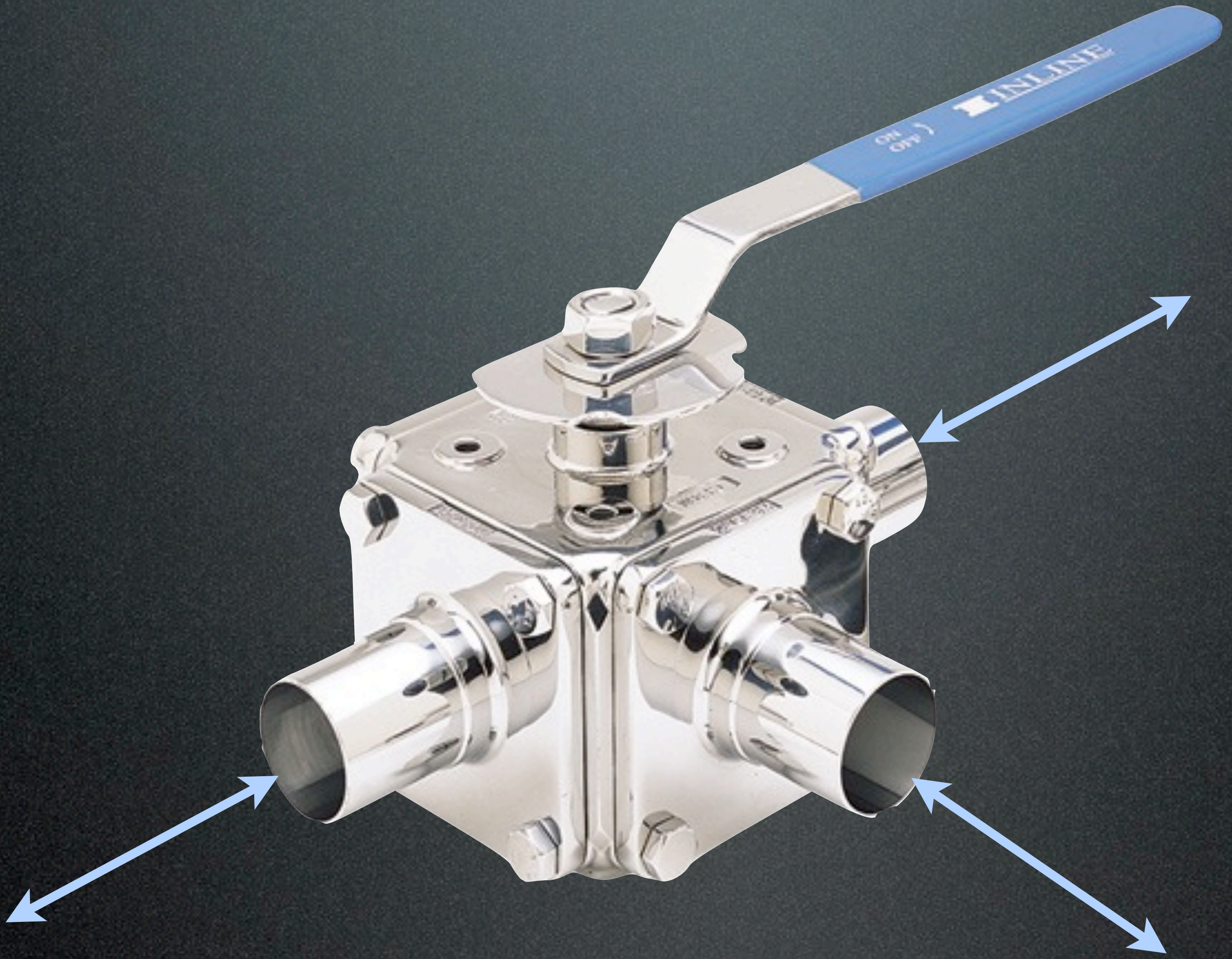
Interoperating with C++ Components

- Our Erlang code interops with C++ code over internal TCP-based protocol
- Marshaling similar to CORBA CDR
- Erlang side implemented as custom behavior
- Features take days to add to C++ side but minutes to implement in Erlang

Web Delivery



Web Delivery

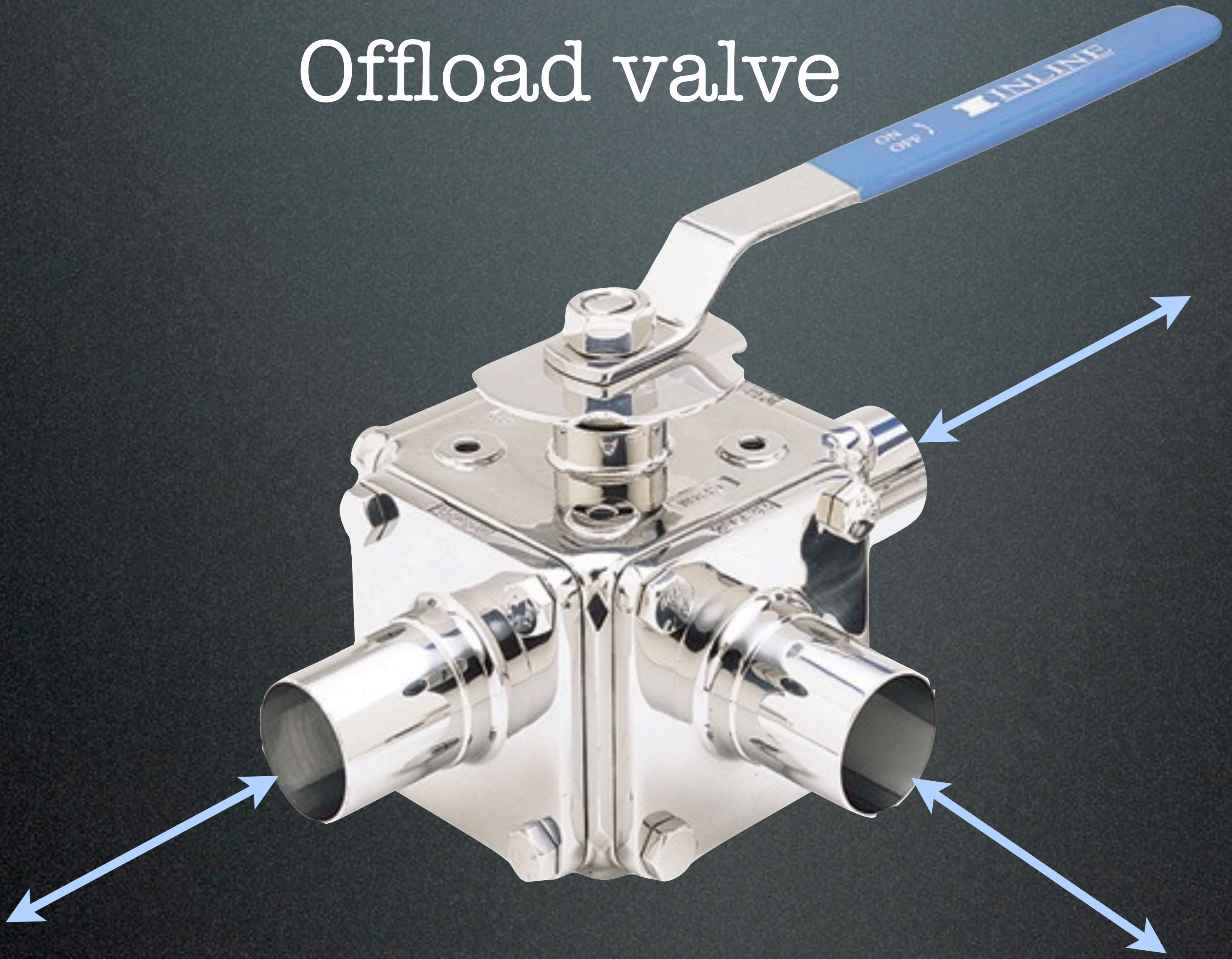


HTTP client

Data plane

Control plane

Web Delivery



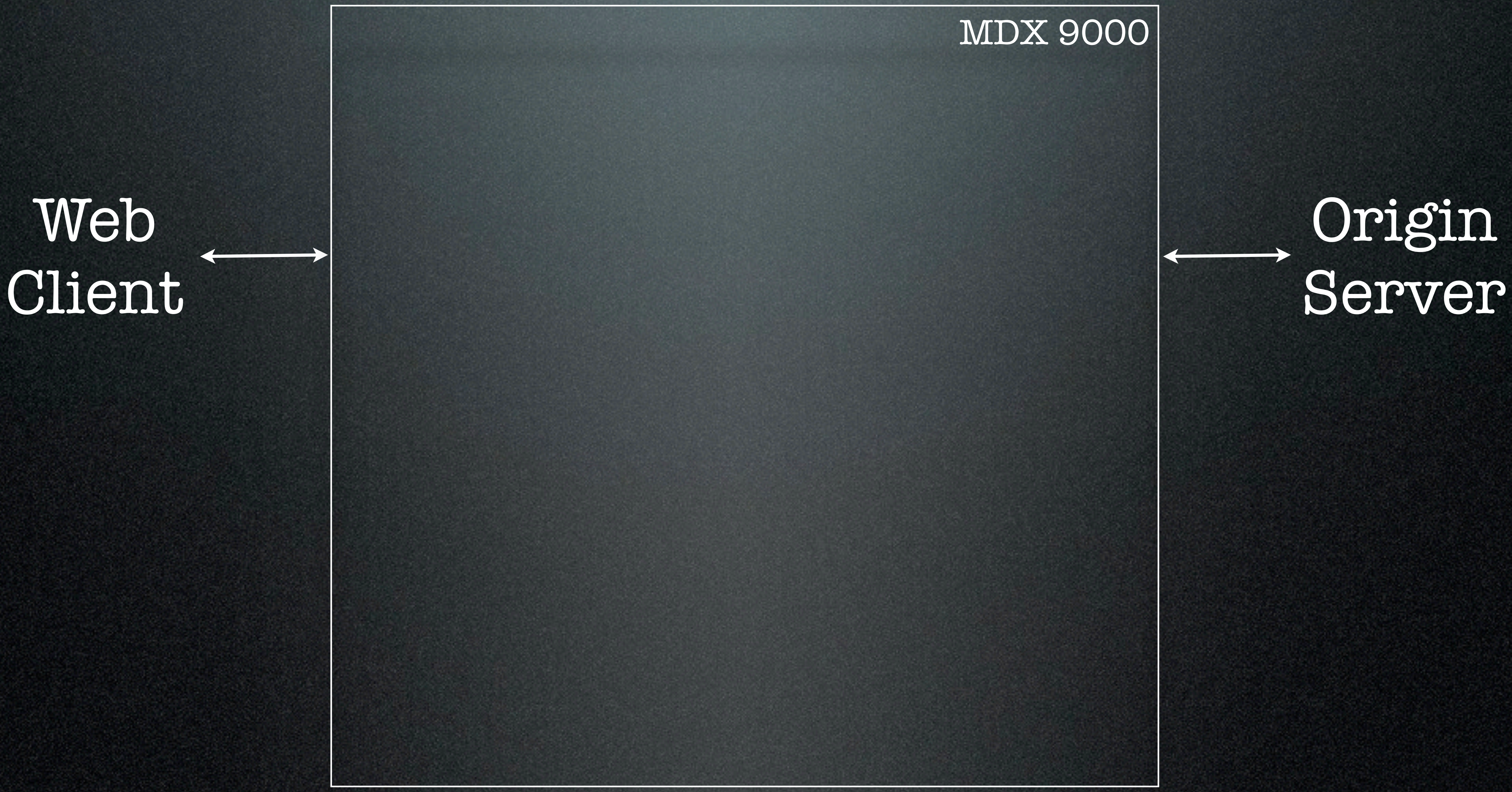
Offload valve

HTTP client

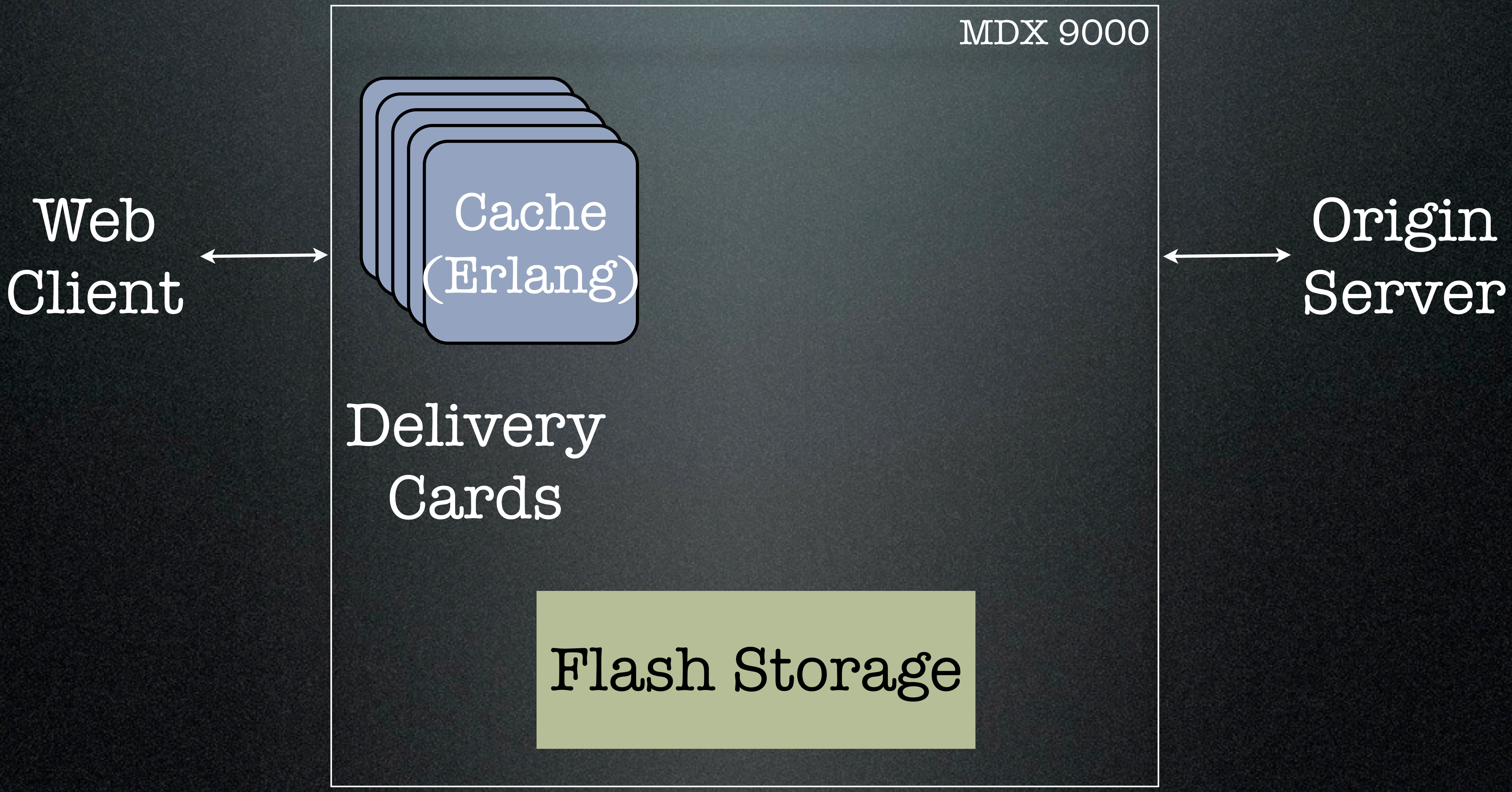
Data plane

Control plane

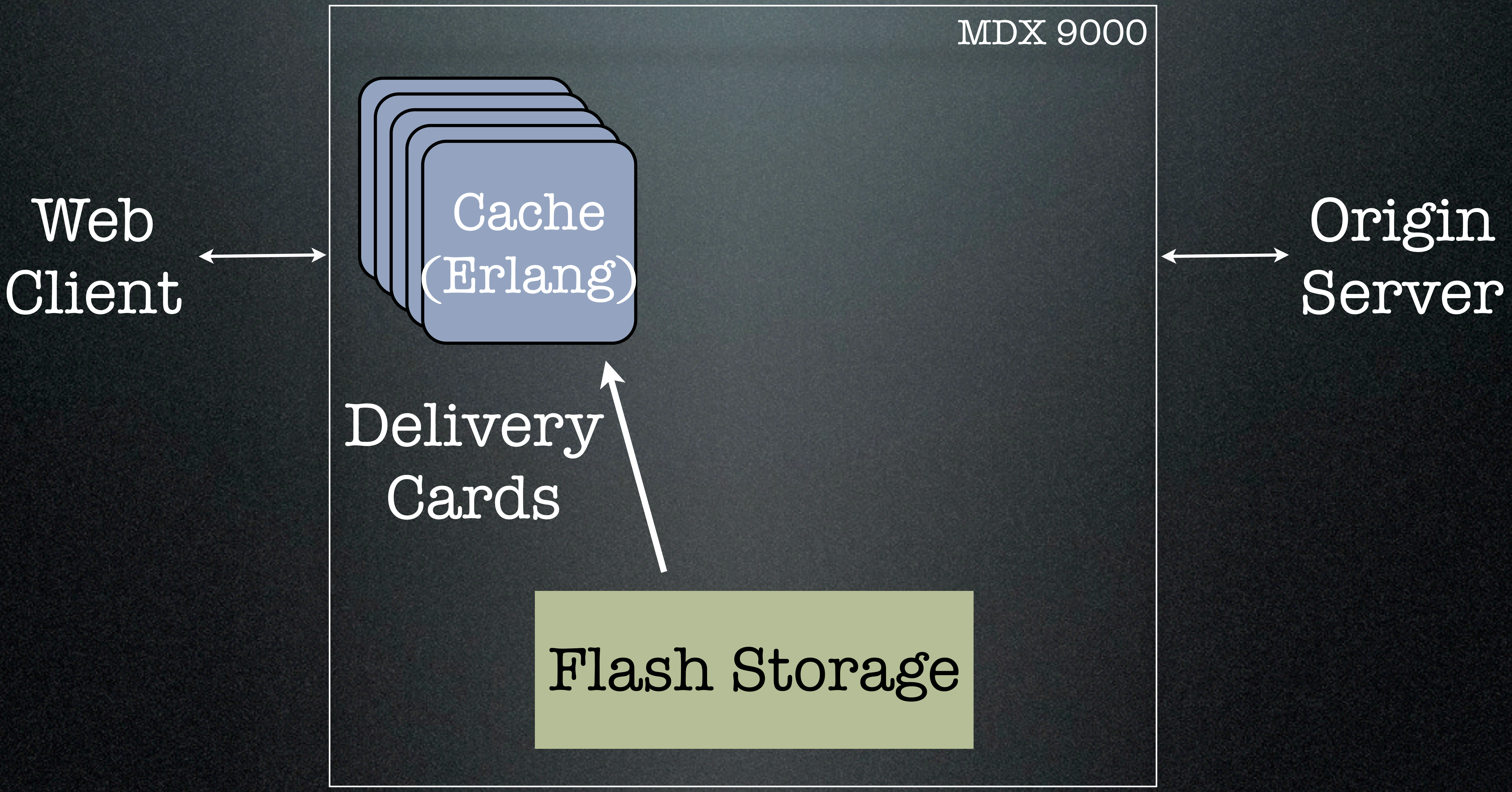
Web Caching



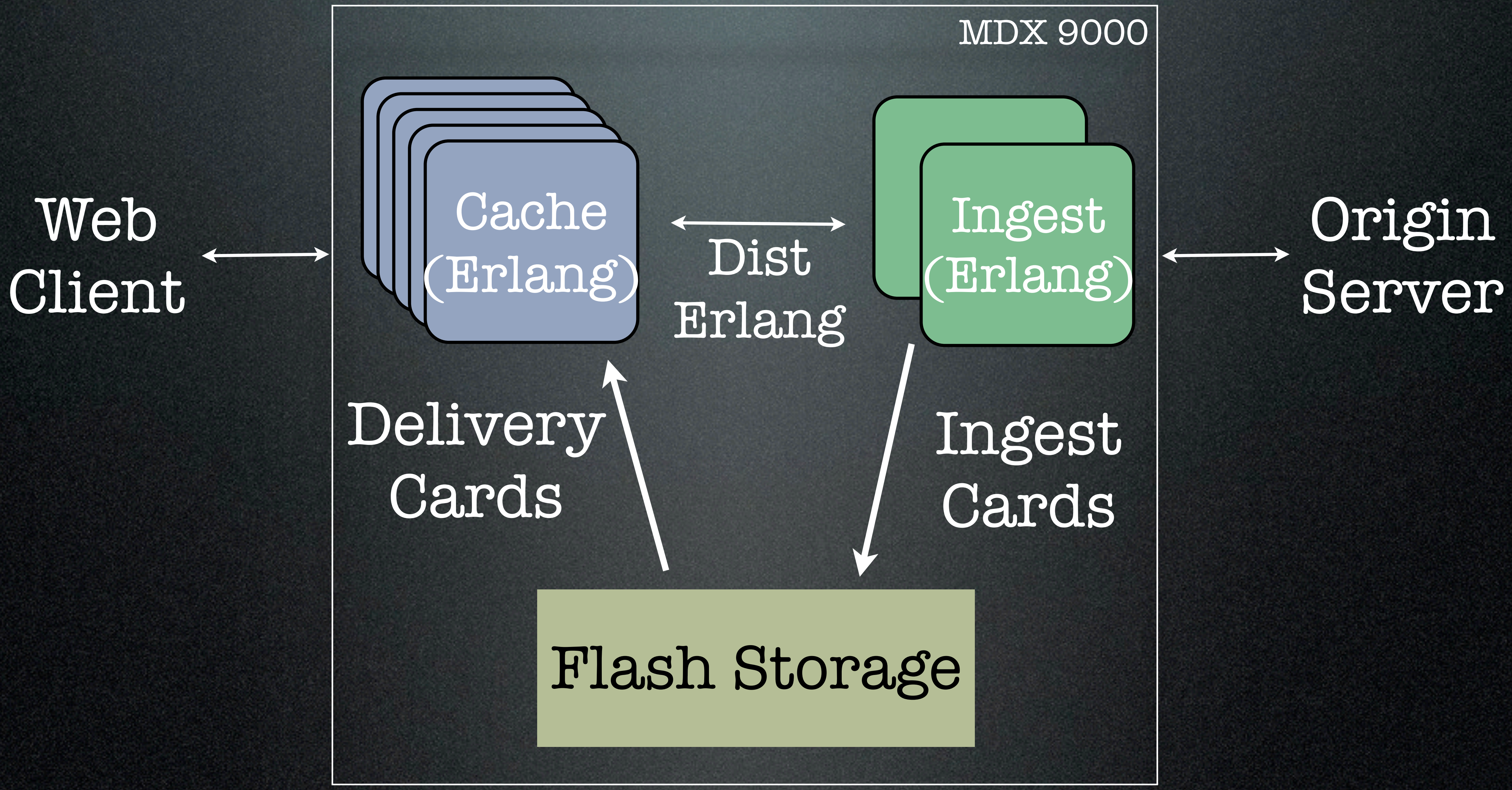
Web Caching


MDX 9000
Web Client
Cache (Erlang)
Delivery Cards
Origin Server
Flash Storage

Web Caching


MDX 9000
Web Client
Cache (Erlang)
Origin Server
Delivery Cards
Flash Storage

Web Caching


MDX 9000
Web Client
Cache (Erlang)
Dist Erlang
Ingest (Erlang)
Origin Server
Delivery Cards
Ingest Cards
Flash Storage

Offload Sockets

- Data pipes to/from flash storage (conceptually)
- Look and act like TCP sockets at the application level
- Implemented in our hardware and software under the covers

Offload Sockets and Erlang

Offload Sockets and Erlang

- Linked-in driver for open, close, and sockopts

Offload Sockets and Erlang

- Linked-in driver for open, close, and sockopts
- Used with gen_tcp via {fd, Fd} option

Offload Sockets and Erlang

- Linked-in driver for open, close, and sockopts
- Used with gen_tcp via {fd, Fd} option
- Tracked by controlling process

Offload Sockets and Erlang

- Linked-in driver for open, close, and sockopts
- Used with gen_tcp via {fd, Fd} option
- Tracked by controlling process
- Yaws integration via fdsrv and streaming API (we make zero changes to Yaws in order to use it)

Advice for New Erlang Users

Unit Testing

Unit Testing

- In a mixed language environment, fingers point at the underdog languages when things go wrong

Unit Testing

- In a mixed language environment, fingers point at the underdog languages when things go wrong
- Due to Fear, Uncertainty, and Doubt

Unit Testing

- In a mixed language environment, fingers point at the underdog languages when things go wrong
- Due to Fear, Uncertainty, and Doubt
- Have to ensure Erlang code is solid

Unit Testing

- In a mixed language environment, fingers point at the underdog languages when things go wrong
- Due to Fear, Uncertainty, and Doubt
- Have to ensure Erlang code is solid
- Developed our own unit testing framework before eunit hit the scene

Unit Testing

- In a mixed language environment, fingers point at the underdog languages when things go wrong
- Due to Fear, Uncertainty, and Doubt
- Have to ensure Erlang code is solid
- Developed our own unit testing framework before eunit hit the scene
- But you should use eunit (we're switching)

Control and Observe

- Module under test has two kinds of dependencies:
  - Function calls – ***M:F(A)***
  - Messages – ***Pid ! Message***
- Control/observe for messages is trivial
- Control/observe for function calls more difficult

Mock Modules

- Used to replace module dependencies at test time
- Typically supply control and observation functions of their own

1. Set up expected values and order of functions to be called
2. Run test
3. Verify things happened as planned

Our Approach

Our Approach

- Read the beam of the module to be mocked

Our Approach

- Read the beam of the module to be mocked
- Generate a module in Abstract Form that provides the same exported funs/arities

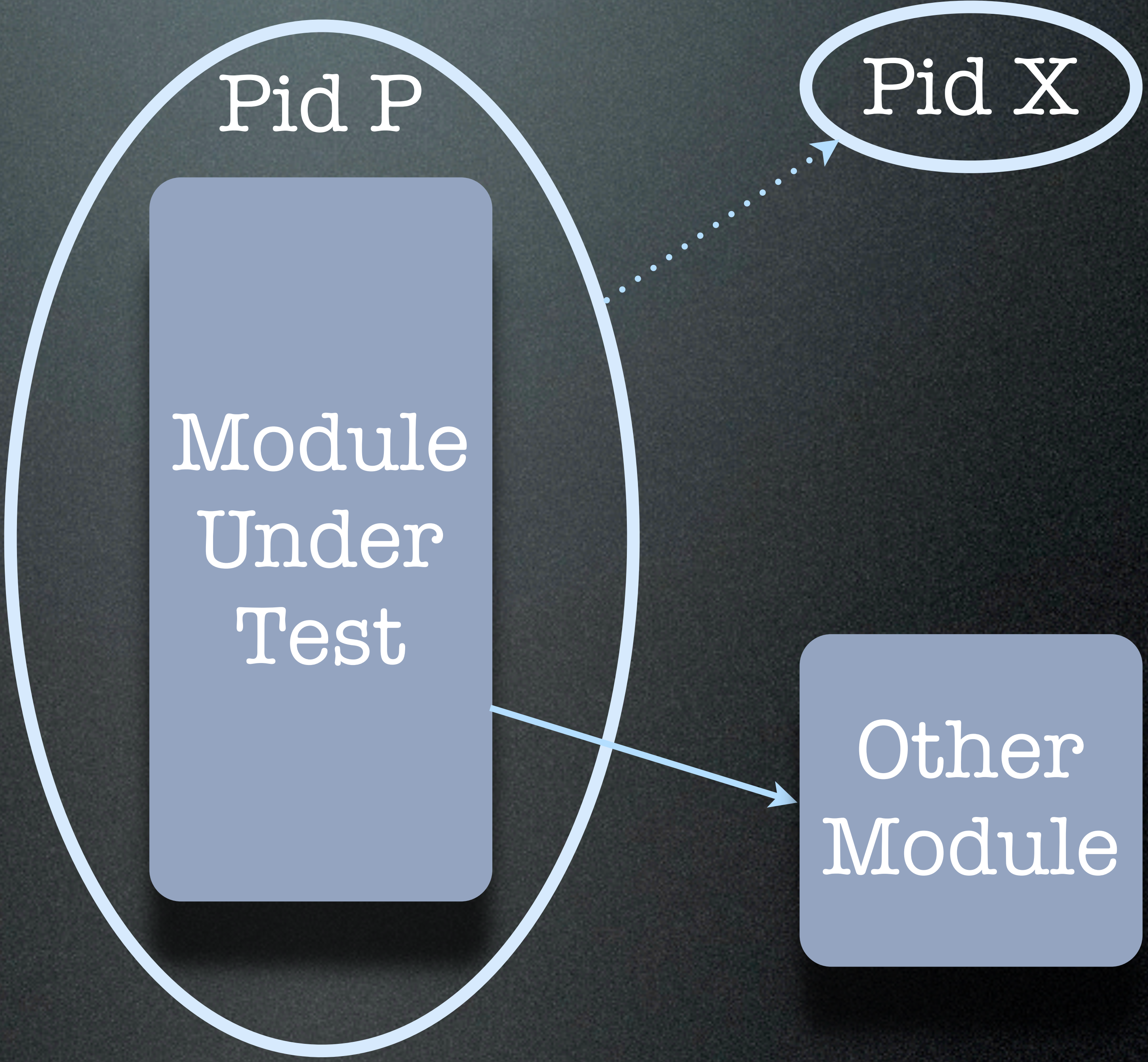
Our Approach

- Read the beam of the module to be mocked
- Generate a module in Abstract Form that provides the same exported funs/arities
- Each fun calls the same-named fun/arity in a specified mock module, usually the test driver module

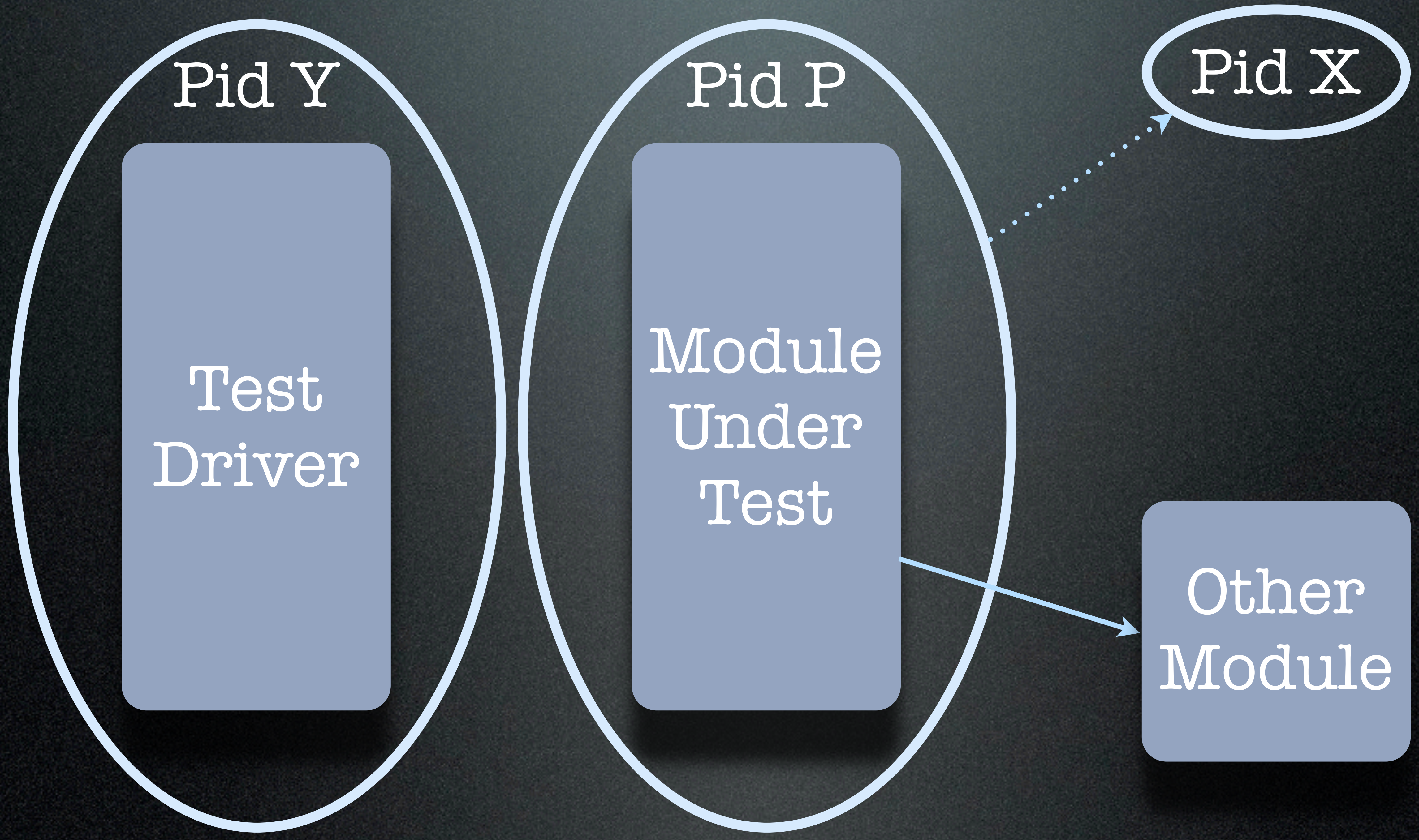
Control and Observe

Pid P
Module Under Test

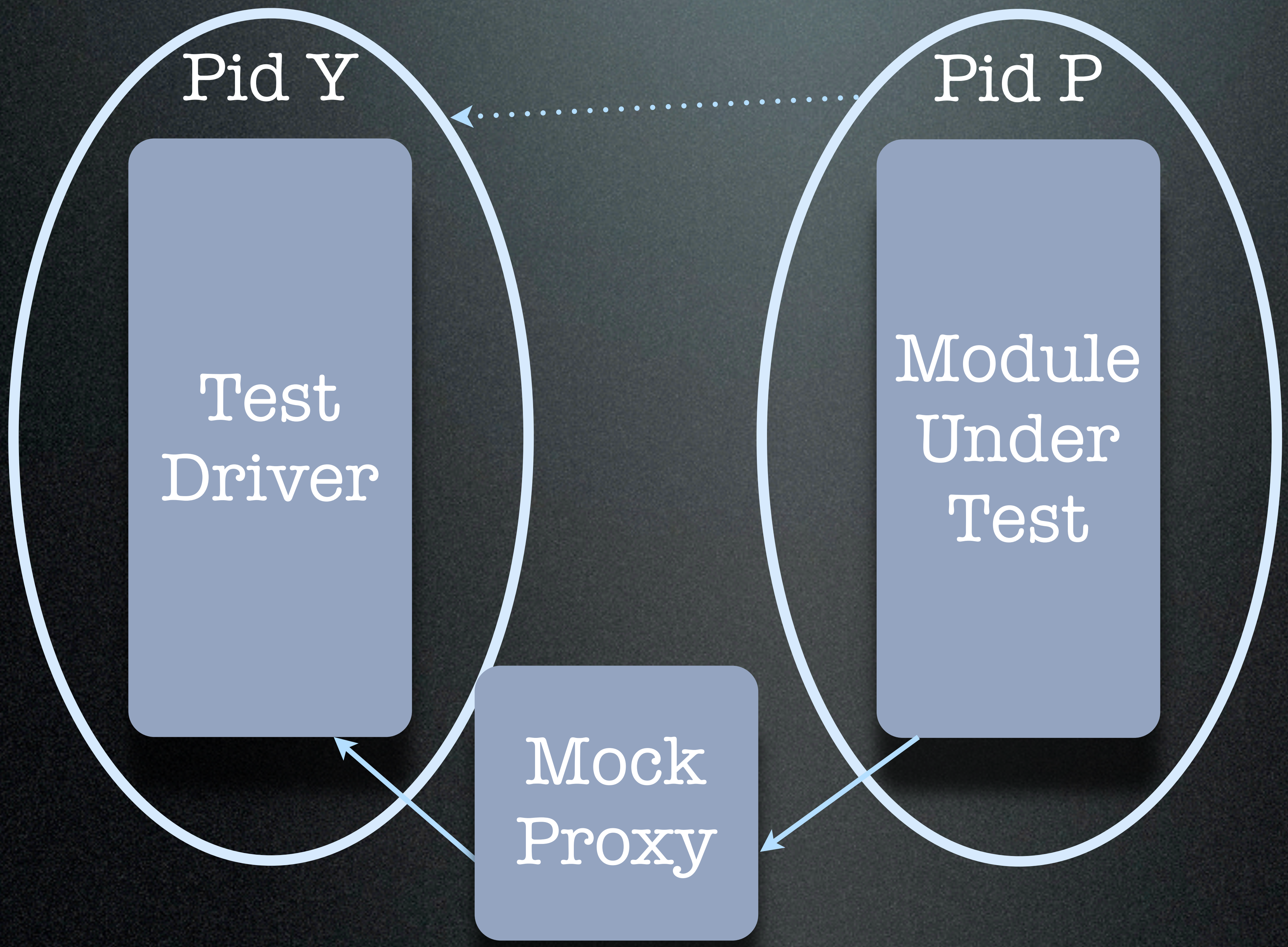
Control and Observe



Control and Observe



Control and Observe



Mock Proxies

- Easily generated, compiled, and loaded at runtime
- Based on Abstract Form “code is data, data is code”
- Multiple proxies can point to a single mock module (usually test driver)
- Allows easy coordination across mocks

Observing with Tracing

- Erlang's built-in tracing allows a designated process to receive events about function calls and messages sends
- This can further enhance observability of module under test
- Eliminates the need for extra control/ observation functions on mock modules

dbg

- Erlang's tracing is one of its most amazing features
- Learn the dbg module, you'll use it every day
- I have never, ever used the Erlang debugger, due to dbg

Advice for New Users

Advice for New Users

- All that great stuff you’ve heard about Erlang? It’s true
- Simple concurrency and coordination
- Hot code loading
- Always-available code tracing
- Sound, practical reliability
- Easy integration
- Enables “production prototypes”
- Open source at github.com

Warning: “Let It Crash”

- This philosophy can be hard for non-Erlangers to buy into
- QA sees a crash in the log, they treat it as something bad. Always.
    - explaining it was designed that way doesn’t always fly
- Programmers new to Erlang (or sometimes not so new) always want to try to handle the errors instead

But “Let It Crash” Works

- Crash and recovery is invaluable for early adopter customers
- They keep using the system even if something goes wrong
- Most of the time, they’re unaware of the crash/recovery
- With dbg and hot code loading, you can debug and repair live systems

Shameless Plug

- New “Functional Web” column just published last week, co-authored with Justin Sheehy
- “Developing RESTful Web Services with Webmachine”
- This and all “Functional Web” columns available at http://steve.vinoski.net/

Thanks