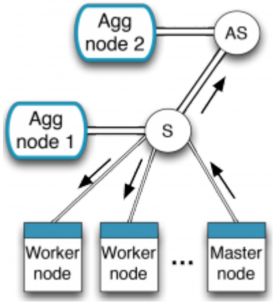
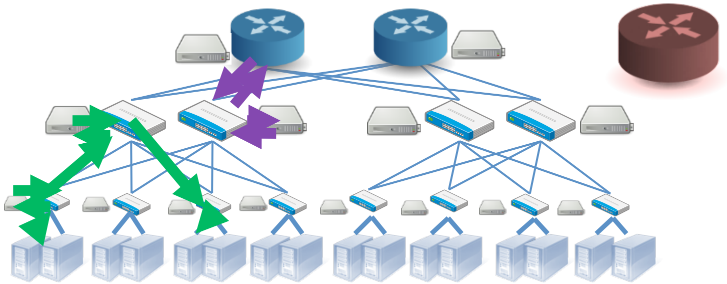




NaaS: Mirage SDN, Switching and Control

University of Cambridge

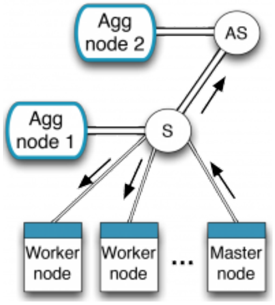
Imperial College London

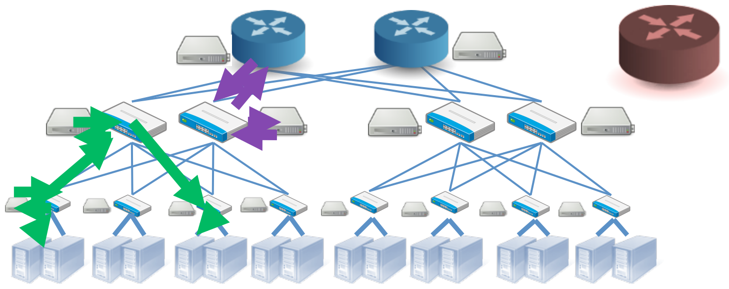
University of Nottingham

Richard Mortier, University of Nottingham^WCambridge
<richard.mortier@cl.cam.ac.uk>

Funded by the UK Engineering and Physical Sciences Research Council (EPSRC)

Challenges for NaaS





1. Performance & efficiency
   - Line rate data processing (10Gbps initially)
   - **Efficient use of network resources**
   - Offload to hardware when possible
2. Programmability & flexibility
   - **Rapid development of new network services**
   - **Simple deployment, resource placement and allocation**
3. Security & safety
   - **Isolation of services within shared hardware**

Current Directions

A key piece of NaaS is ***flexibility****: how can applications better use network resources?*

- OpenFlow Alternatives
  - P4, POF
- Raising the abstraction
  - Above the transport layer
- Self-scaling applications
  - Distributed control

OpenFlow Alternatives

- OpenFlow has limitations:
  - Standard continues to expand (12 to 41 fields)
  - Fixed header fields limits experimentation
  - Extensible match fields provide substrate for extensibility
- Interested in investigating alternatives
  - E.g., **Protocol Oblivious Forwarding** (Huawei), **P4** (Bosshart et al, http://arxiv.org/pdf/1312.1719.pdf)

Emphasising Extensibility

- POF (Huawei)
  - Multi-level table matching
  - (Pointer, Offset), pointer guaranteed to advance
- P4 (Barefoot, Intel, Stanford, MSR, Google, Princeton)
  - "Parse and populate" model
  - Programmable parser, parallel matching
  - Actions composed of protocol independent switch primitives

How do these alternatives compare, and are they sufficient for our needs?

Raising the Abstraction

- OpenFlow focuses on packets and flows
  - The usual Ethernet and IP headers (incl. IPv6)
  - Plus some support for key control protocols (ARP, DHCP, MPLS, VLAN)
  - Plus statistics messages
- But applications deal in Layer 8 behaviour
  - Mirage *applications* link SDN libraries directly
  - How will applications use them?

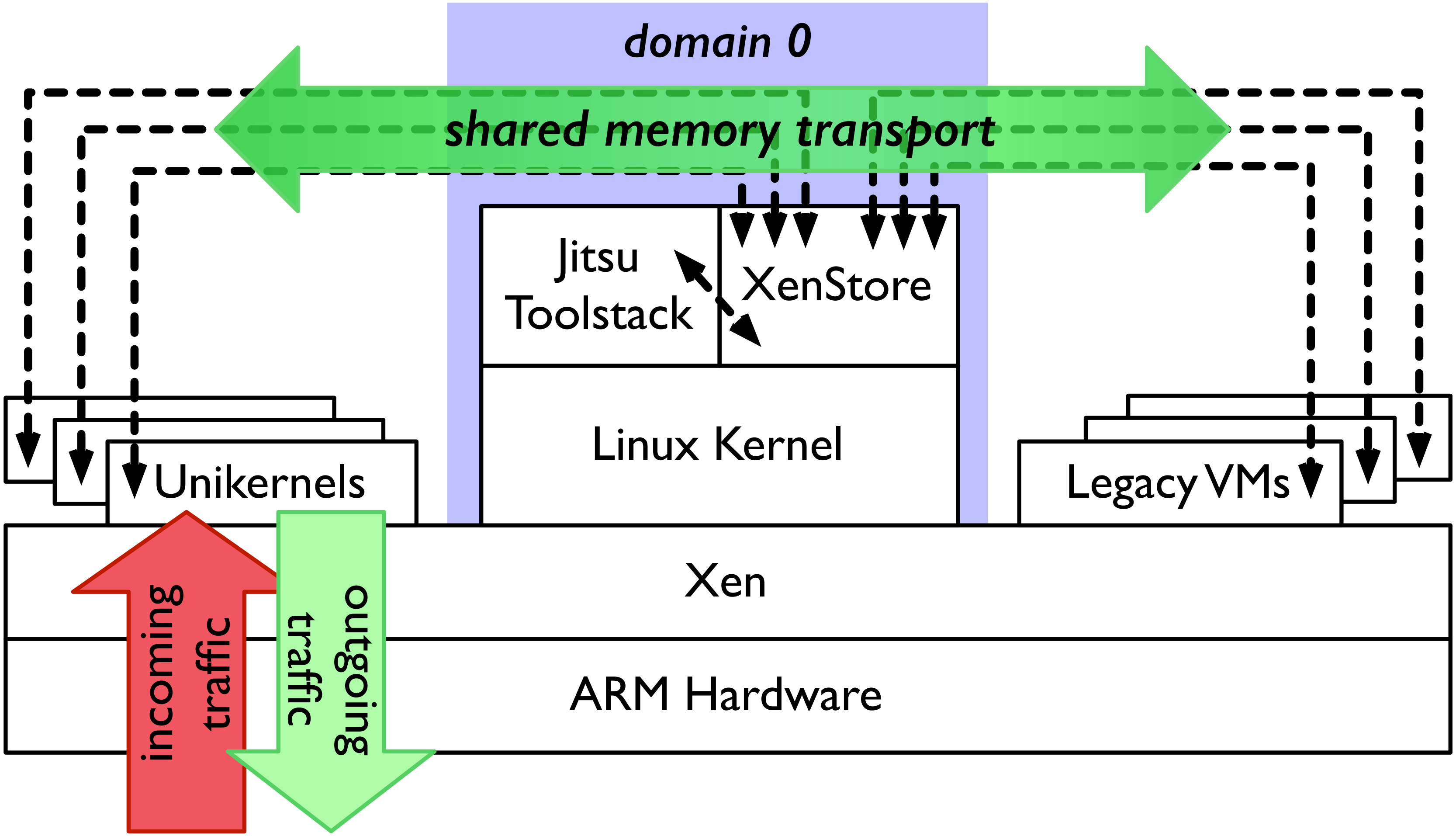
How can we raise abstractions so that applications can deal in layer 8 concepts (e.g., URLs)?

Self-Scaling Applications

- Use of the network by applications is rather naïve
  - The network as a black box
  - Little to no direct control
- SDN allows us to go further
  - Applications install handlers to deal with particular sets of network traffic

How can we take this further, enabling application logic to modify its own deployment – e.g., scaling up/down in response to load – in light of network conditions?

Scaling Out: Jitsu



domain 0

shared memory transport

Jitsu Toolstack

XenStore

Linux Kernel

Unikernels

Legacy VMs

Xen

ARM Hardware

incoming traffic

outgoing traffic

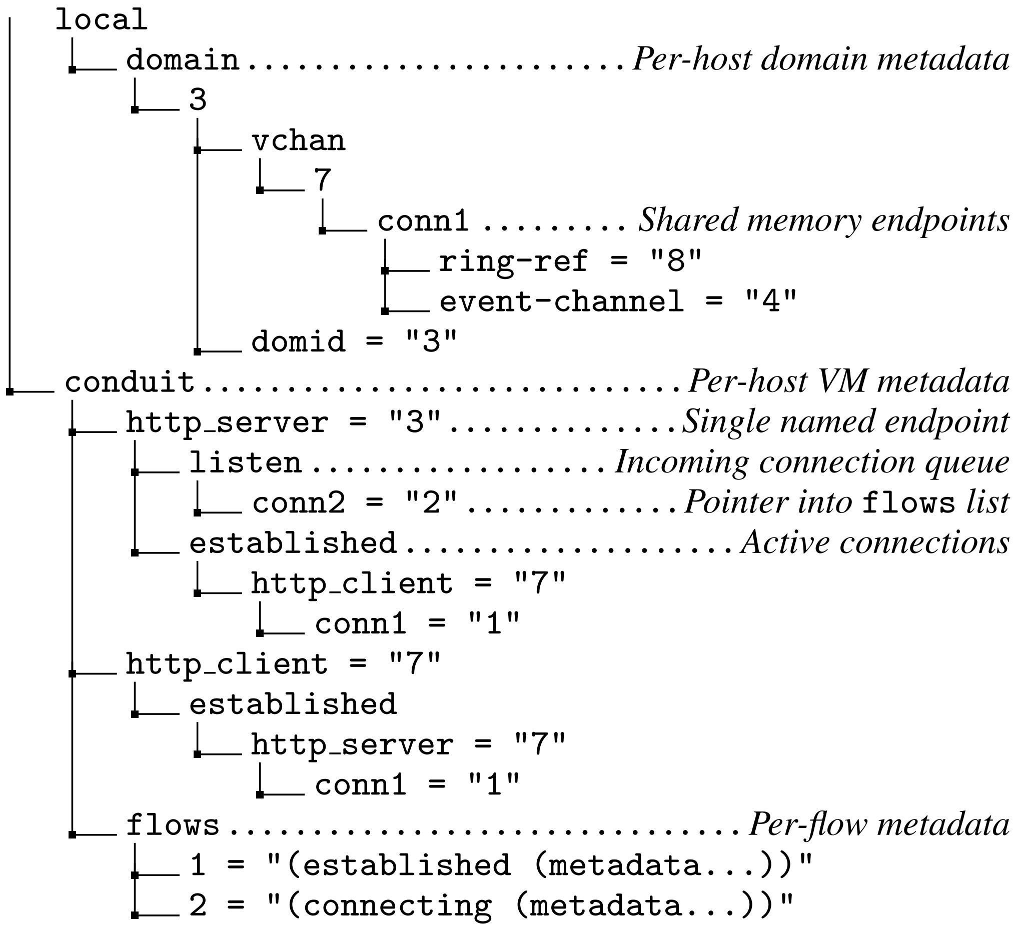
Conduit: Efficient Inter-VM Comms

Zero-copy shared-memory pages between peers

- Xen grant tables map pages between VMs, synchronised via event channels
- Rendezvous facility so VMs discover named peers
- Supports unikernel and legacy VM rendezvous
- Hooks into higher-level name services like DNS

- Compatible with the *vchan* inter-VM communication protocol

Conduit: XenStore Layout

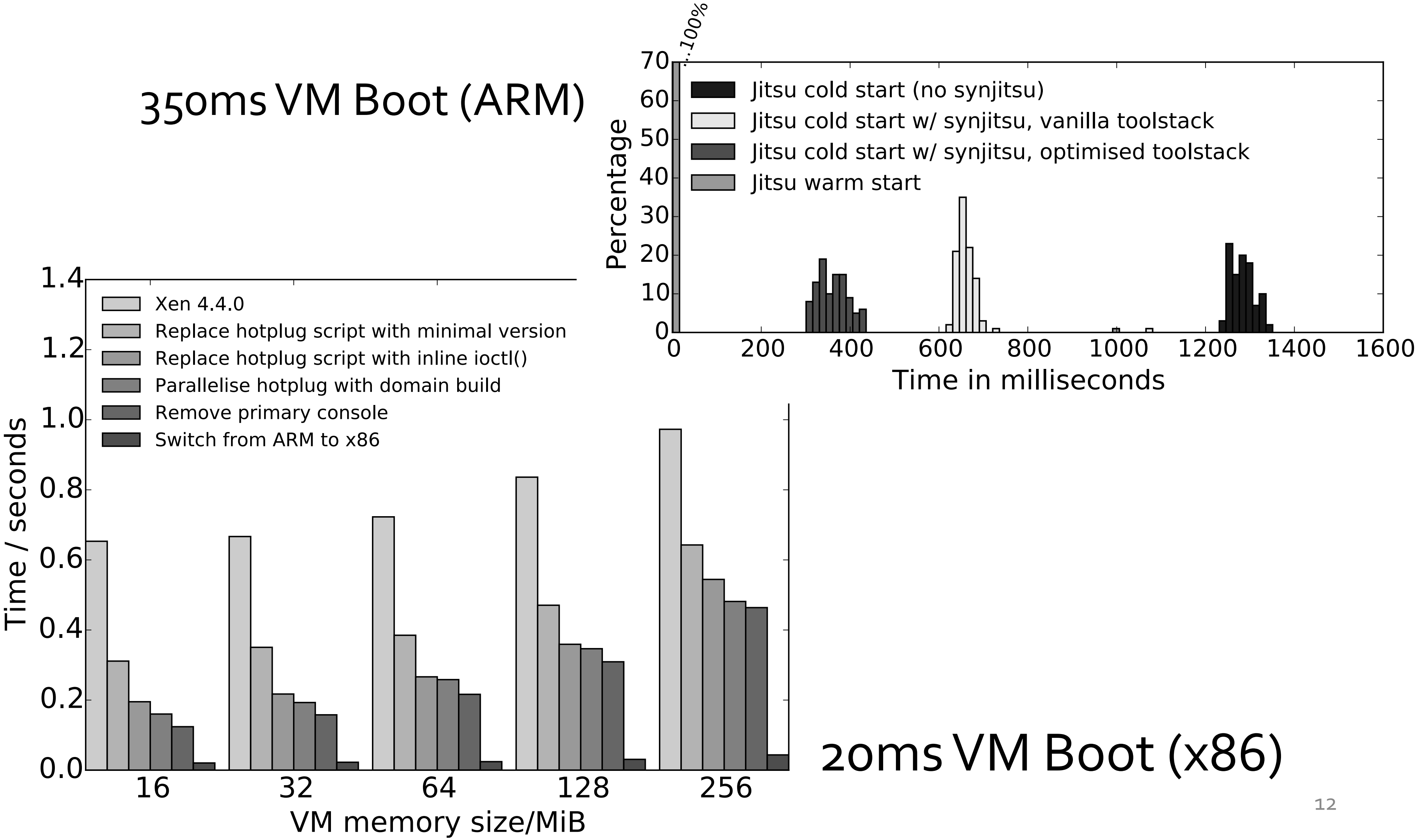


```
local
├── domain ...................... Per-host domain metadata
│   └── 3
│       ├── vchan
│       │   └── 7
│       │       └── conn1 ......... Shared memory endpoints
│       │           ├── ring-ref = "8"
│       │           └── event-channel = "4"
│       └── domid = "3"
└── conduit .......................... Per-host VM metadata
    ├── http_server = "3" ............. Single named endpoint
    │   ├── listen ................. Incoming connection queue
    │   │   └── conn2 = "2" ............ Pointer into flows list
    │   └── established .................. Active connections
    │       └── http_client = "7"
    │           └── conn1 = "1"
    ├── http_client = "7"
    │   └── established
    │       └── http_server = "7"
    │           └── conn1 = "1"
    └── flows ........................... Per-flow metadata
        ├── 1 = "(established (metadata...))"
        └── 2 = "(connecting (metadata...))"
```

Jitsu Directory Service

- Performs the role of Unix's *inetd*:
  - Jitsu VM launches at boot time to handle name resolution (whether local via a well known jitsud Conduit node in XenStore or remote via DNS)
  - When a request arrives for a live unikernel, Jitsu returns the appropriate endpoint
  - If the unikernel is not live, Jitsu boots it, and acts as proxy until the unikernel is ready

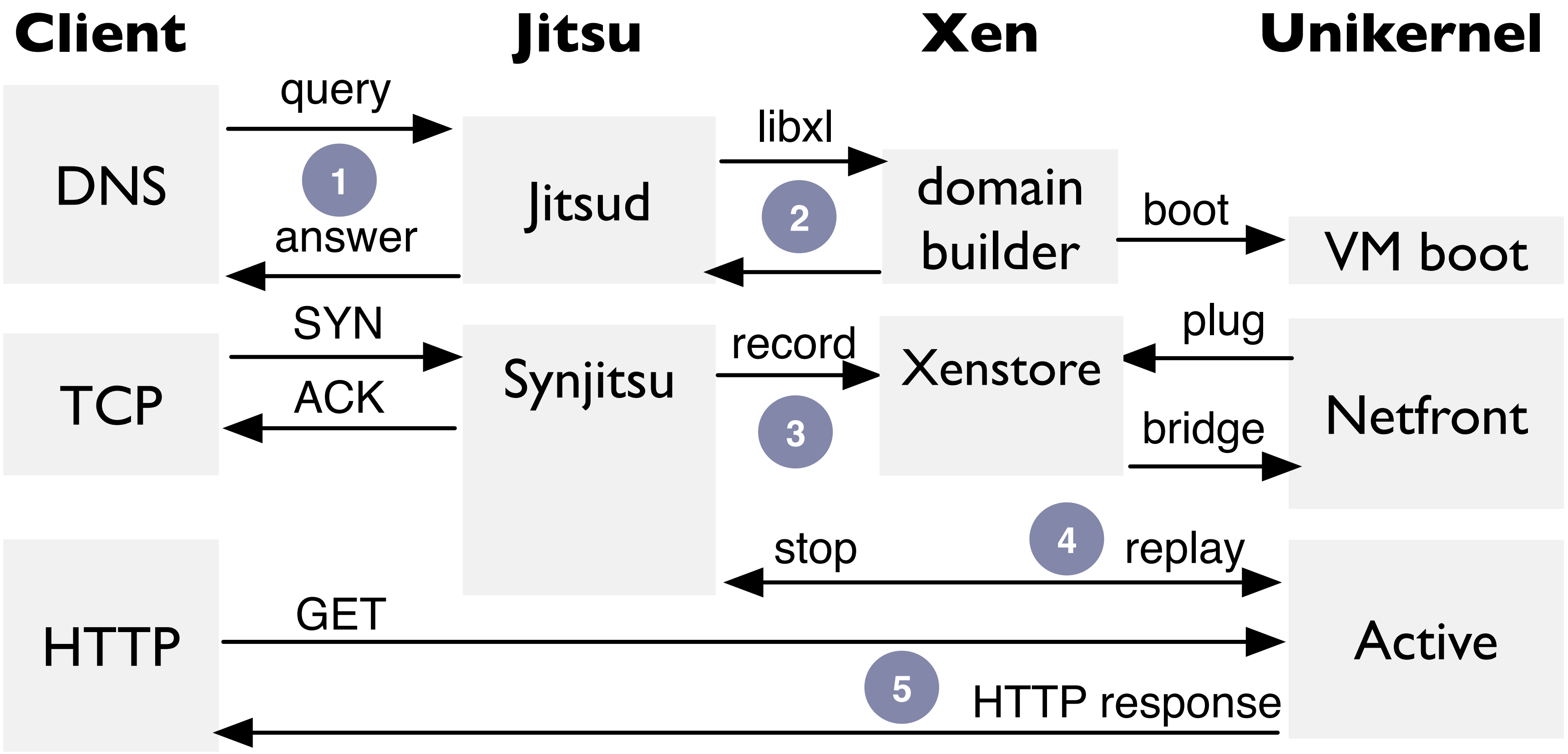
Result: Low Latency Boot



350ms VM Boot (ARM)

20ms VM Boot (x86)

Synjitsu: Masking Latency



By buffering TCP requests into XenStore and then replaying, Synjitsu parallelises connection setup and unikernel boot

Summary

So current themes are focused on investigating several ways to improve flexibility

- Replacing OpenFlow
- Raising the level of abstraction
- Automating datacenter deployment