Lecture 1 : Edge Computing

Dr. Bibhas Ghoshal

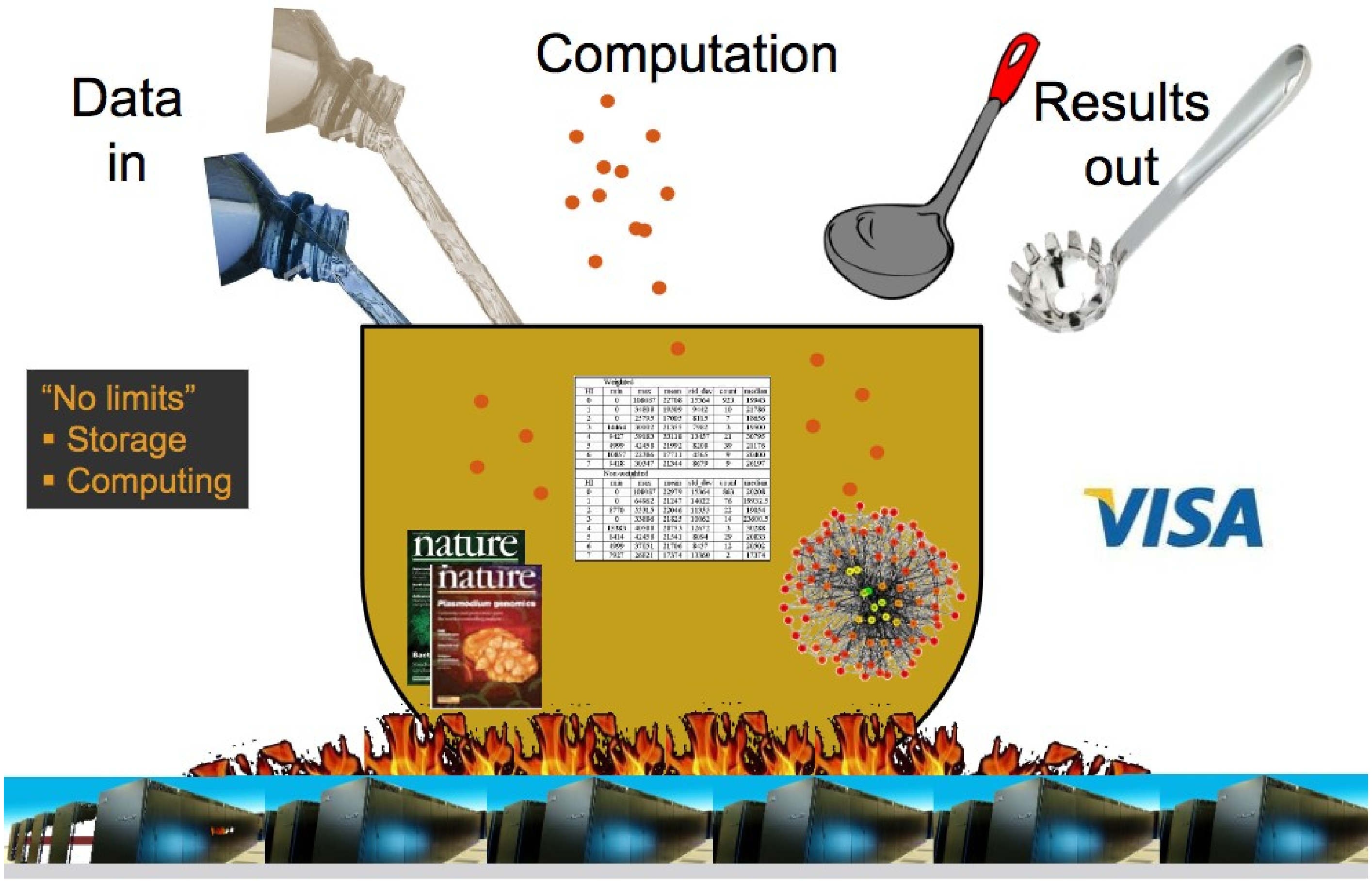
Assistant Professor

Department of Information Technology

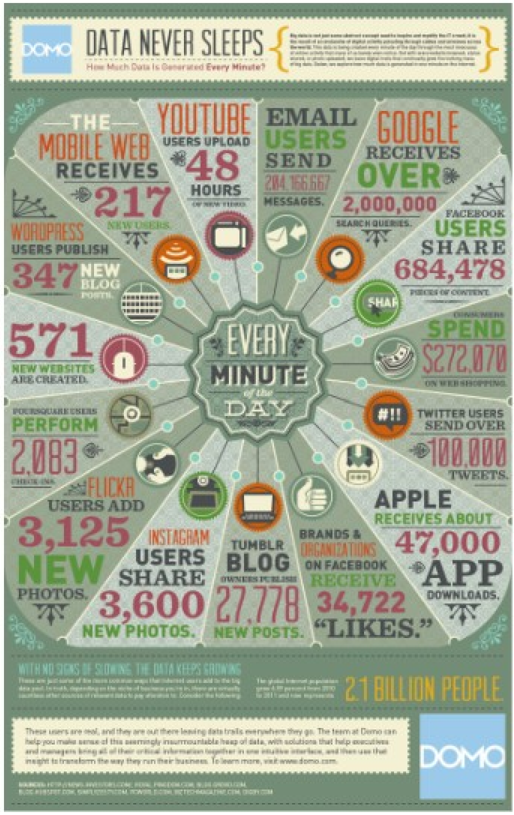
Indian Institute of Information Technology Allahabad

Cloud Computing

The “Standard” Cloud



Cloud Computing



Key observation: much data originates not in the data center but at the "edges"

100s of zettabytes

Mobile web

Sensors

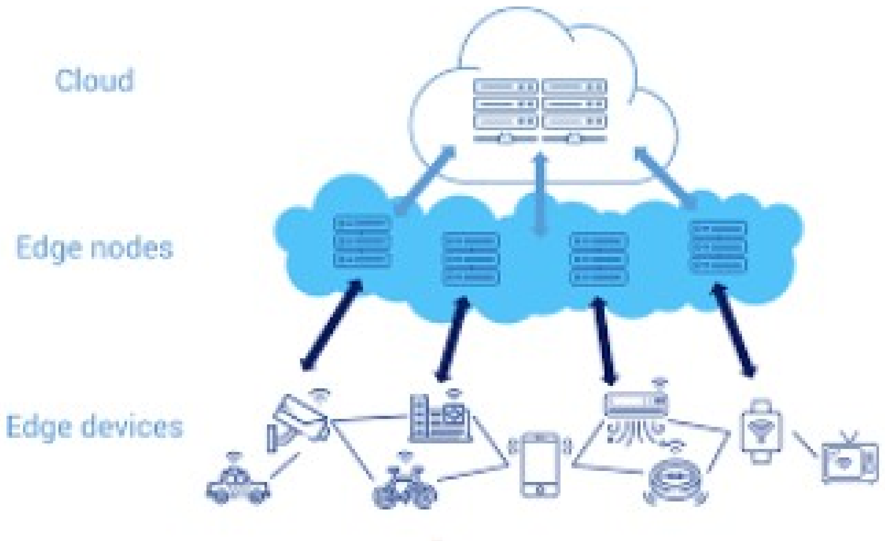
IoT devices

Automobiles

Etc.

Edge Computing

Far edge: sensor/IoT, human
– very limited networking
• The “edge”
– local compute, storage
– 1 hop to far edge, Internet connected
• Local cloud
– Collection of edge nodes
• Centralized cloud



Edge Computing

far-edge (data gen: sensors, actuators, cars, robots, human)
-> near-edge (carried: phones, tablets, wearables)
-> localized (infra: one-hop server)
-> micro-DC (infra: close-by resources, fog)
-> geo-distributed-edge (infra: WAN)
-> central-cloud (infra: WAN)

- Notion of far/near edge, localized/micro-DC may be blurry

Why the Shift?

Centralization => Dispersion

- Reasons
  - Proximity/latency: highly responsive cloud services/applications (e.g. AR, VR, cognitive assistance)

- Low latency, high b/w, low jitter
  - Scalability via edge analytics

- Local processing of high b/w sensors (e.g. cameras, cars)
  - Privacy enforcement

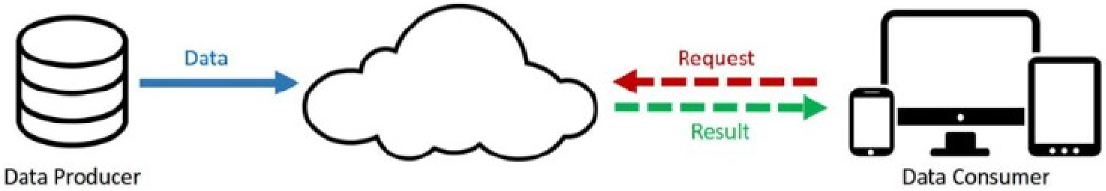
- First point of contact between far edge and system
  - Masking cloud outages
  - Sheer volume of edge resources and far edge IoT devices (~ 50 billion things)

Edge Evolution

Why Now?

Networking: SDN/NFV, Ultra-low-latency, 5/6G

- Computing power: smartphones, wearables, etc.
- Explosion of data at the edge



From Just a Data Producer to Data Consumer and Producer

Edge Computing Models

Mobile offloading: face recognition (lat and energy)

- Cloud offloading: shopping cart updating (lat)
- Edge data processing: localized data analytics

– Local search (e.g. lost child) => lat
– Filtering (e.g. remove faces) => privacy, b/w
Aggregation (e.g. combine data) =>b/w

Challenges

Technical :

All the usual problems with dispersion
– reliability, naming, programming, naming, heterogeneity, scalability

- Algorithms, systems for collective control and sharing of edge resources
- Runtime infrastructure: edge services
- Complexity management
- Weaker security perimeter

Non-Technical :

- Edge infrastructure: who provides it?
- Edge business case: who funds it?

Cloud Offloading

Rich, interactive applications are emerging in mobile context

- Apple’s Siri, AR apps
- Wearable devices

Cloud offloading

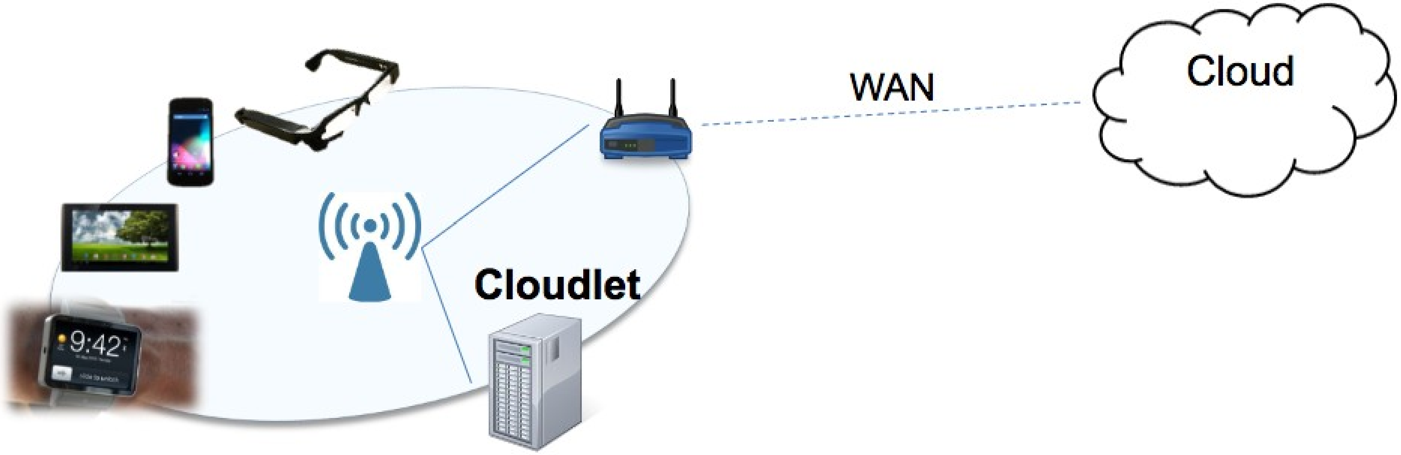
- These applications are too expensive to run on clients alone!
- Offload computation to a back-end server at cloud

Today’s remote cloud is a suboptimal place; high latency and limited bandwidth

Optimize for user’s attention

Cloudlet as Nearby Offload Site

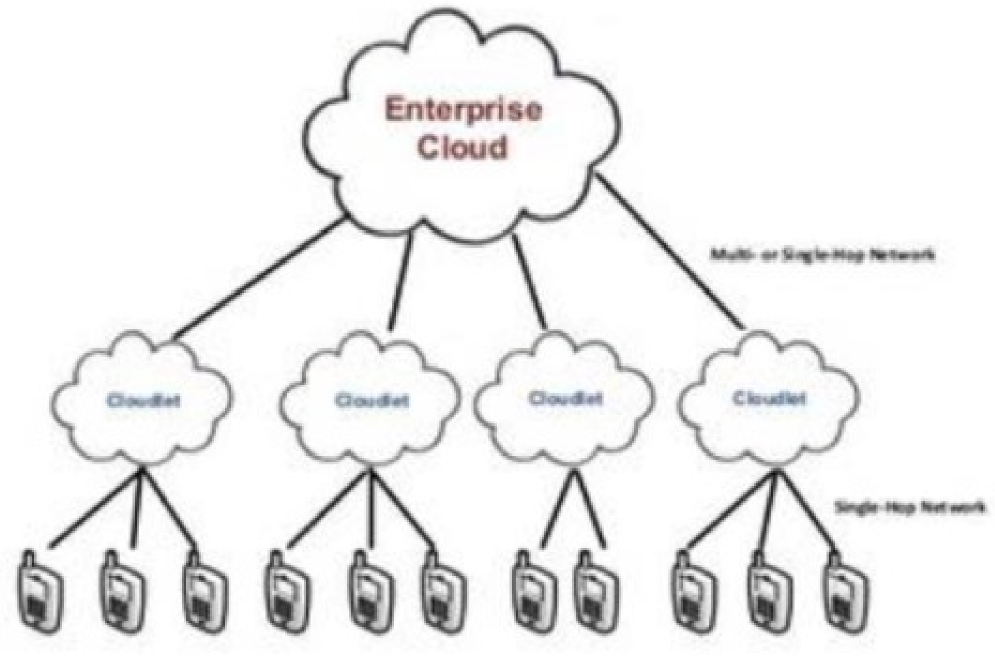
Cloudlet: a nearby offloading site dispersed at the edges of the Internet
-> Let’s bring the cloud closer!



How to launch a custom back-end server at an arbitrary edge?

Cloudlet

Focus on Deployment and Infrastructure



Challenges

- To make this viable and scalable, we need an edge infrastructure (maybe 3 rd party)

– Wide-area: think mobiles and travel
– Shared: multiple apps running on the edge
– Enable any apps in any language in any OS + software libraries, etc.
– Robust
    - Secure
    - Disconnected fallback

- Need to encapsulate apps in VMs
- Granularity?

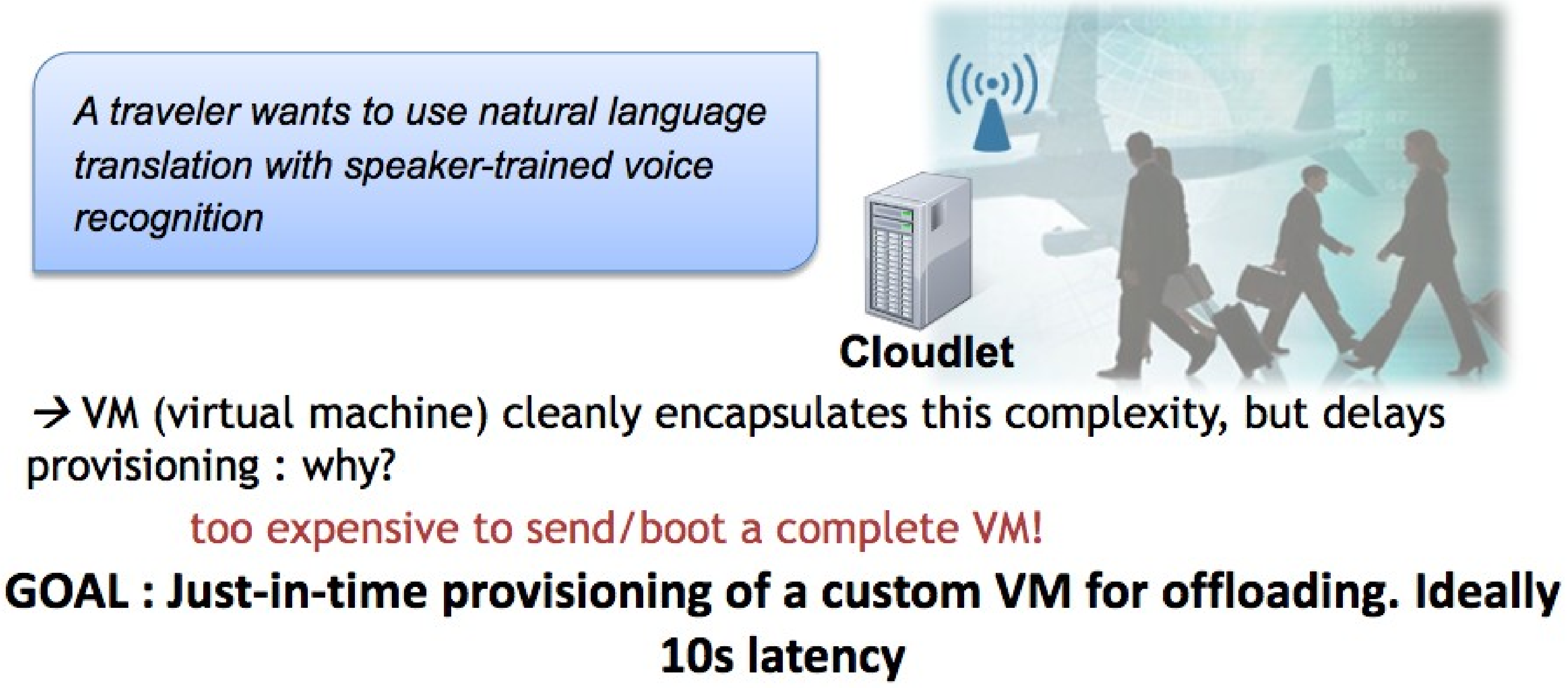
Options

- Static provisioning
  - Store all possible VMs on the edge nodes
  - Feasible?
  - Advantages?

- Dynamic provisioning
  - Issues?

Just In Time Provisioning

1. Support widest range of user customization including OS, language, and library
2. Strong isolation between untrusted computations
3. Access control, metering, dynamic resource management, …



A traveler wants to use natural language translation with speaker-trained voice recognition

→ VM (virtual machine) cleanly encapsulates this complexity, but delays provisioning : why?

too expensive to send/boot a complete VM!

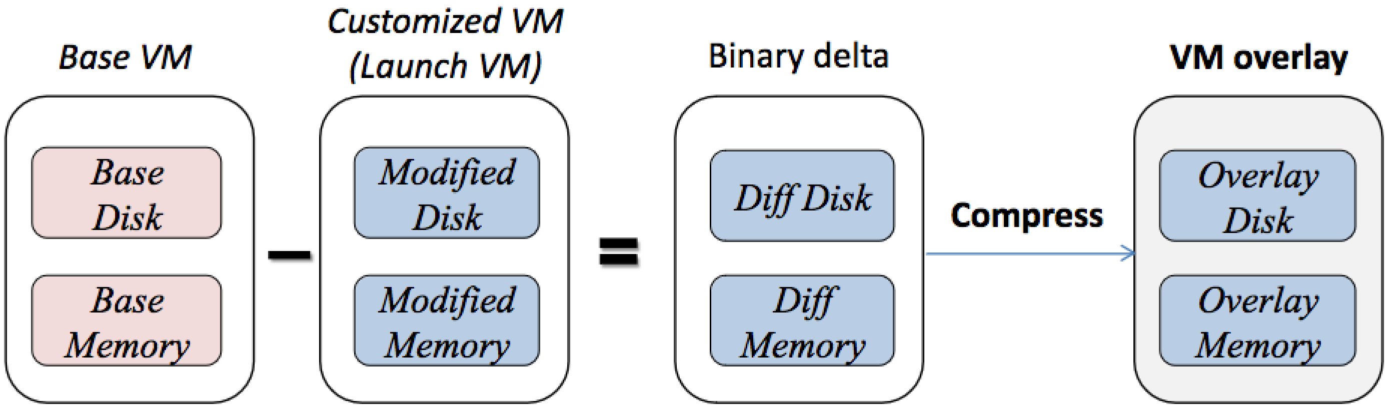
GOAL : Just-in-time provisioning of a custom VM for offloading. Ideally 10s latency

VM Synthesis

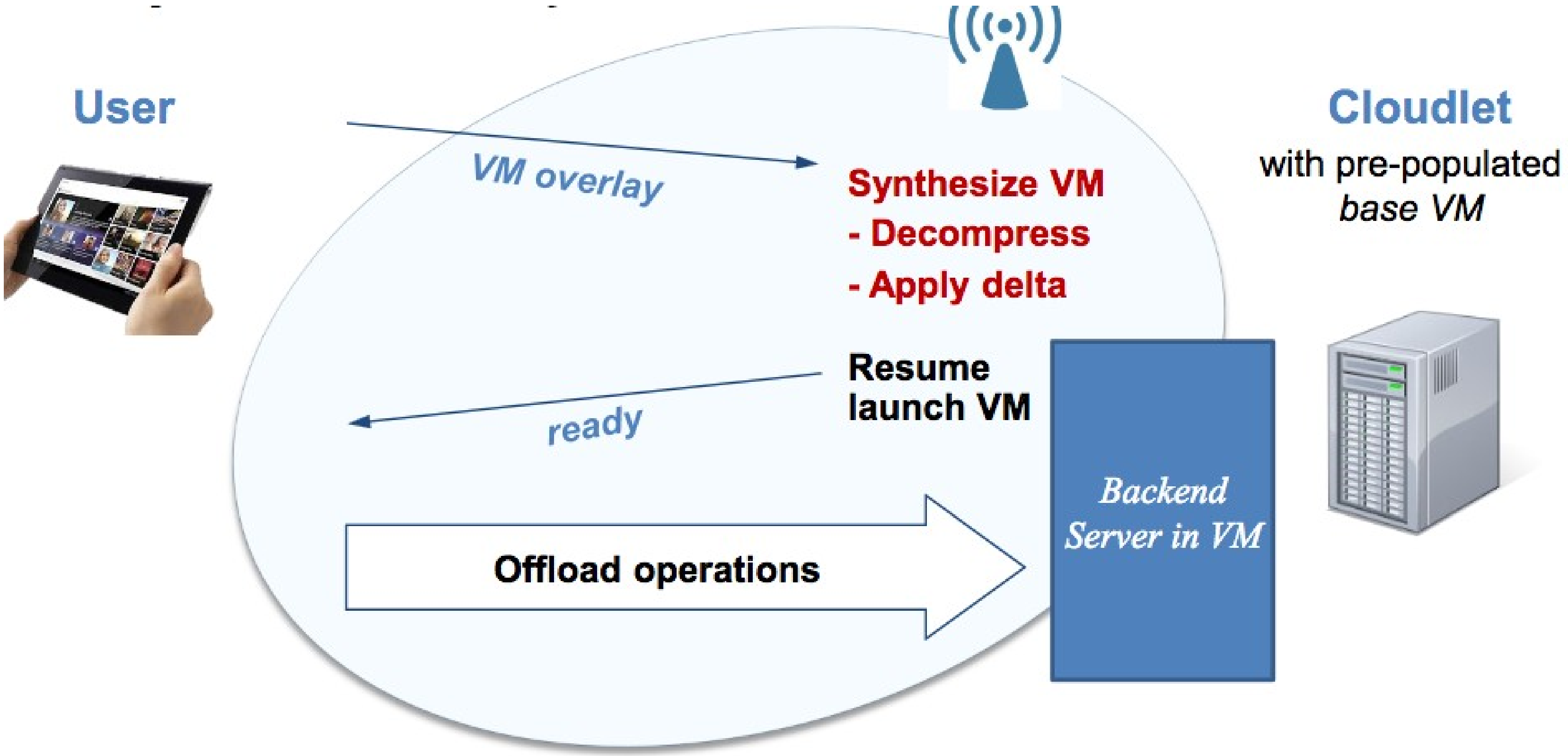
VM Synthesis: dividing a custom VM into two pieces

1) Base VM: Vanilla OS that contains kernel and basic libraries

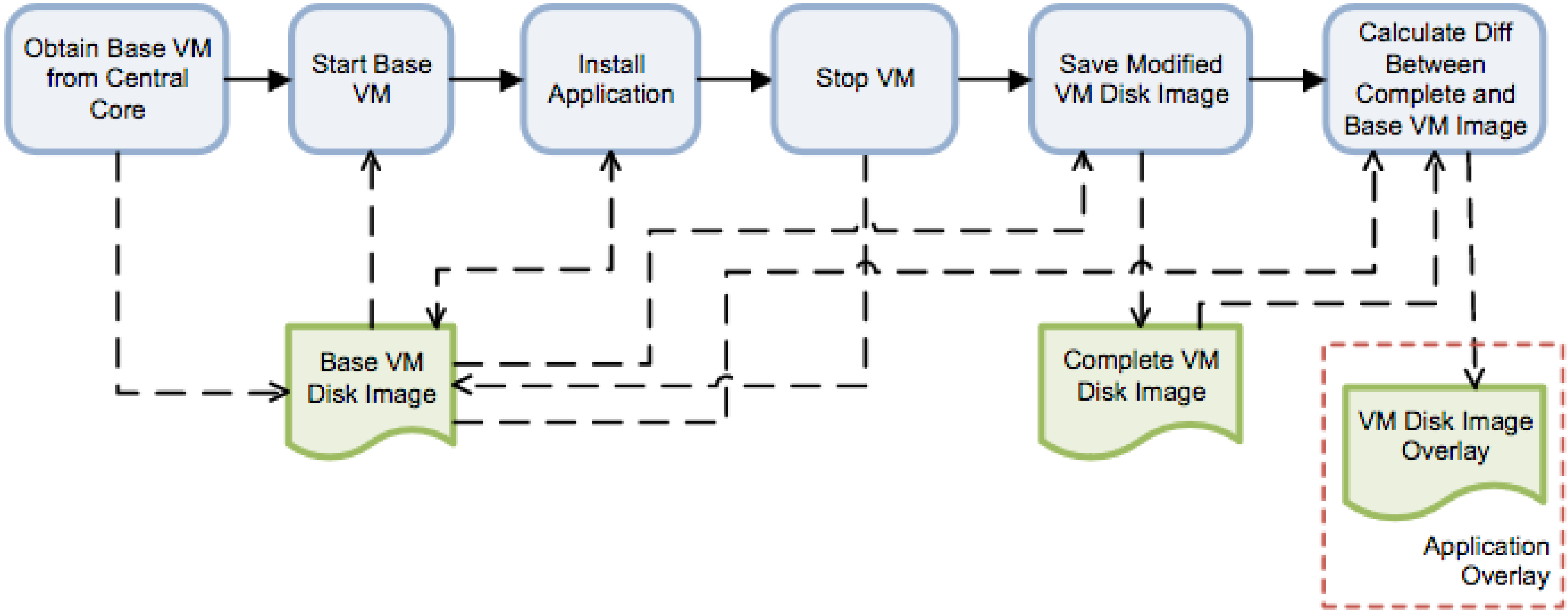
2) VM overlay: A binary patch that contains customized parts



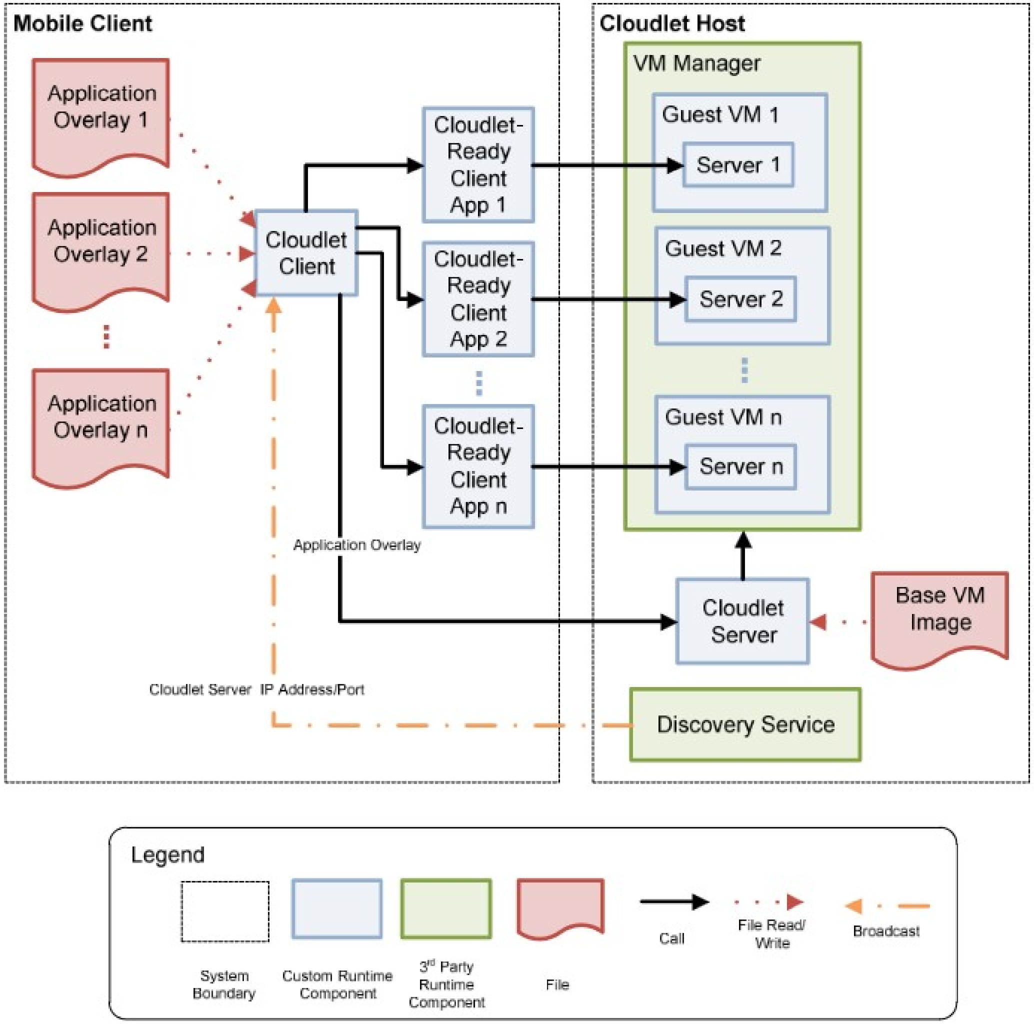
Steps for VM Synthesis



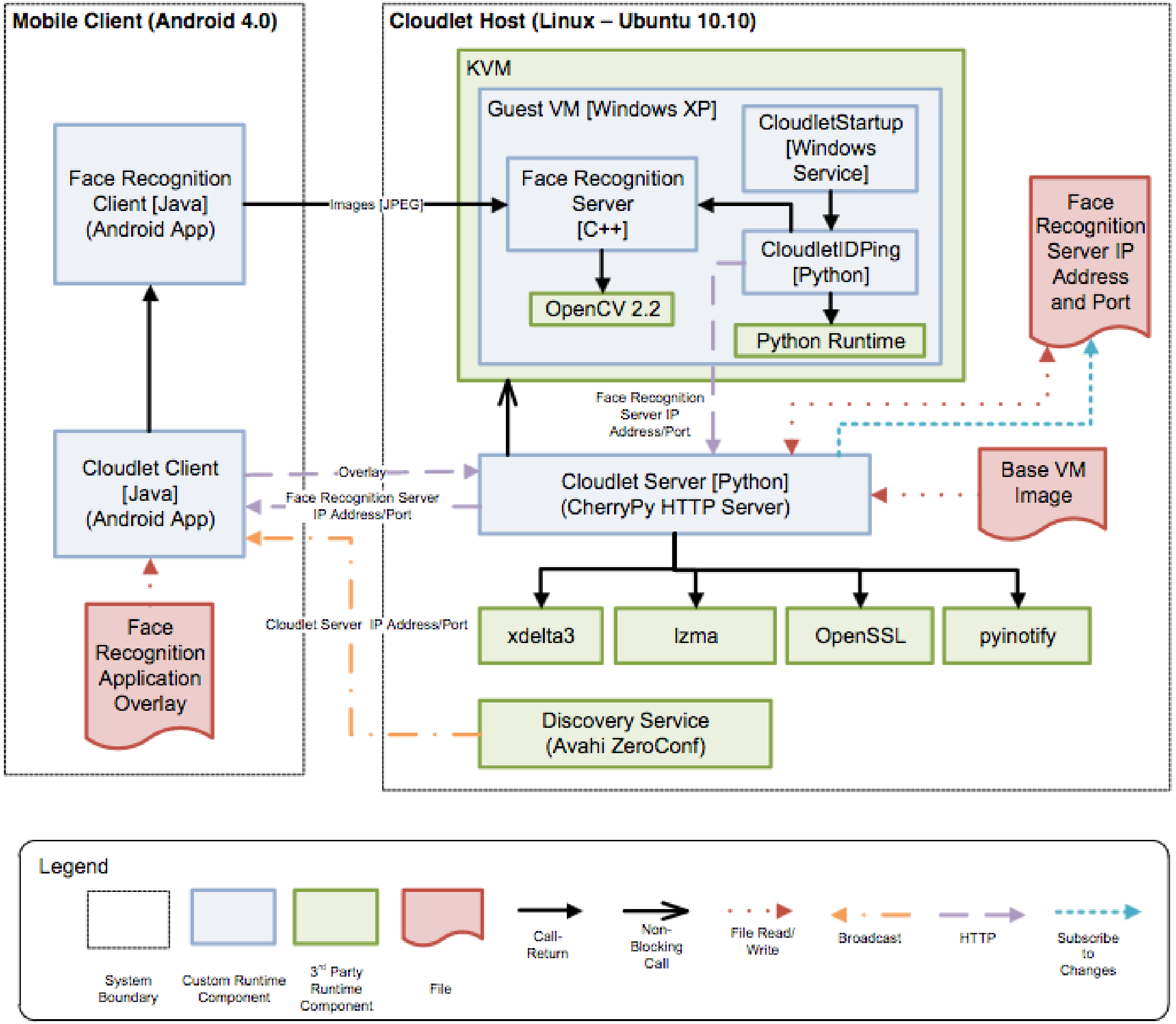
Application Overlay



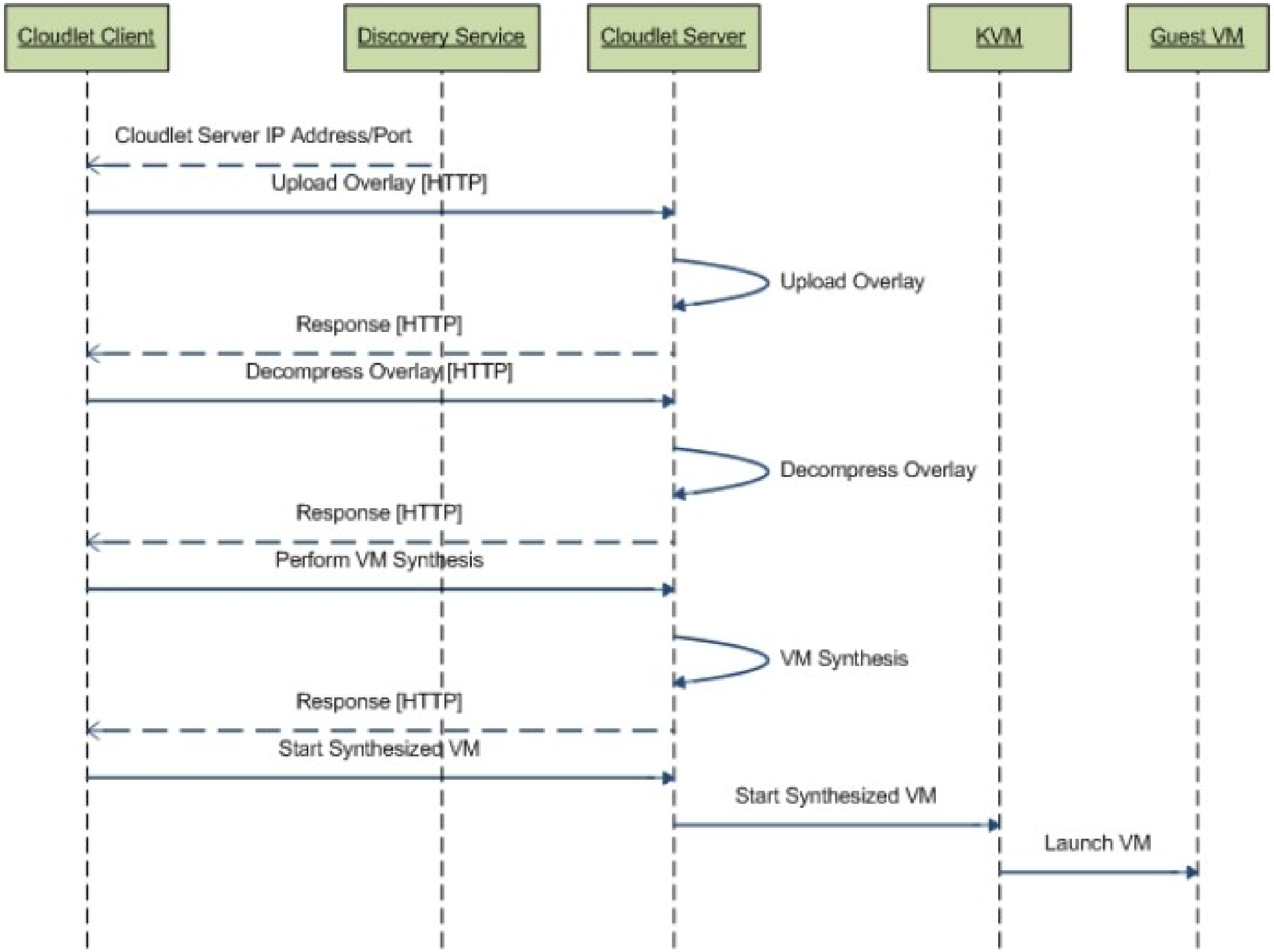
Cloudlet based code offload



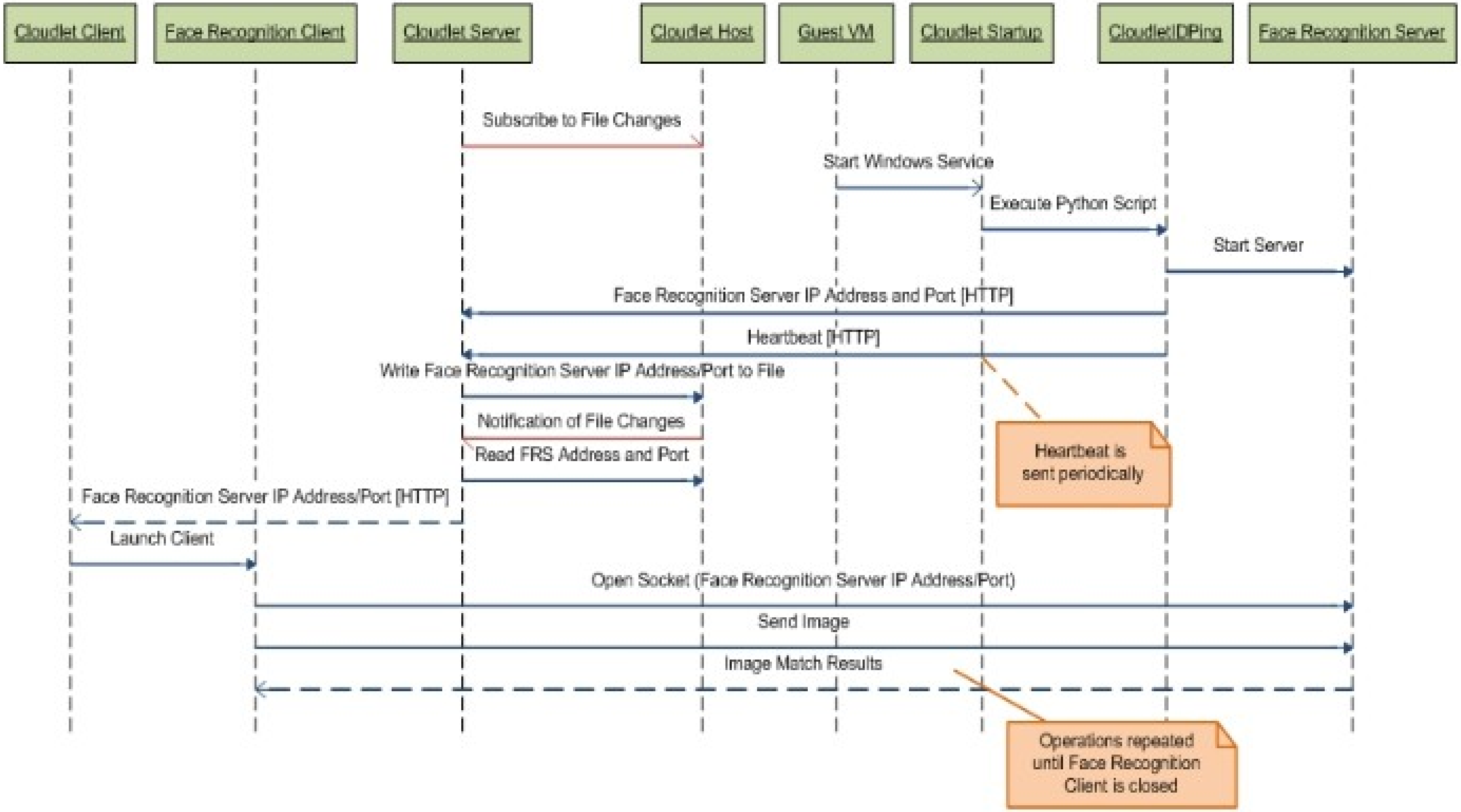
Cloudlet based code offload


Mobile Client (Android 4.0)
Face Recognition Client [Java] (Android App)
Cloudlet Client [Java] (Android App)
Face Recognition Application Overlay
Images [JPEG]
Overlay
Face Recognition Server IP Address/Port
Cloudlet Server IP Address/Port
Cloudlet Host (Linux – Ubuntu 10.10)
KVM
Guest VM [Windows XP]
Face Recognition Server [C++]
CloudletStartup [Windows Service]
CloudletIDPing [Python]
OpenCV 2.2
Python Runtime
Face Recognition Server IP Address and Port
Face Recognition Server IP Address/Port
Cloudlet Server [Python] (CherryPy HTTP Server)
Base VM Image
xdelta3
lzma
OpenSSL
pyinotify
Discovery Service (Avahi ZeroConf)
Legend
System Boundary
Custom Runtime Component
3rd Party Runtime Component
File
Call-Return
Non-Blocking Call
File Read/ Write
Broadcast
HTTP
Subscribe to Changes

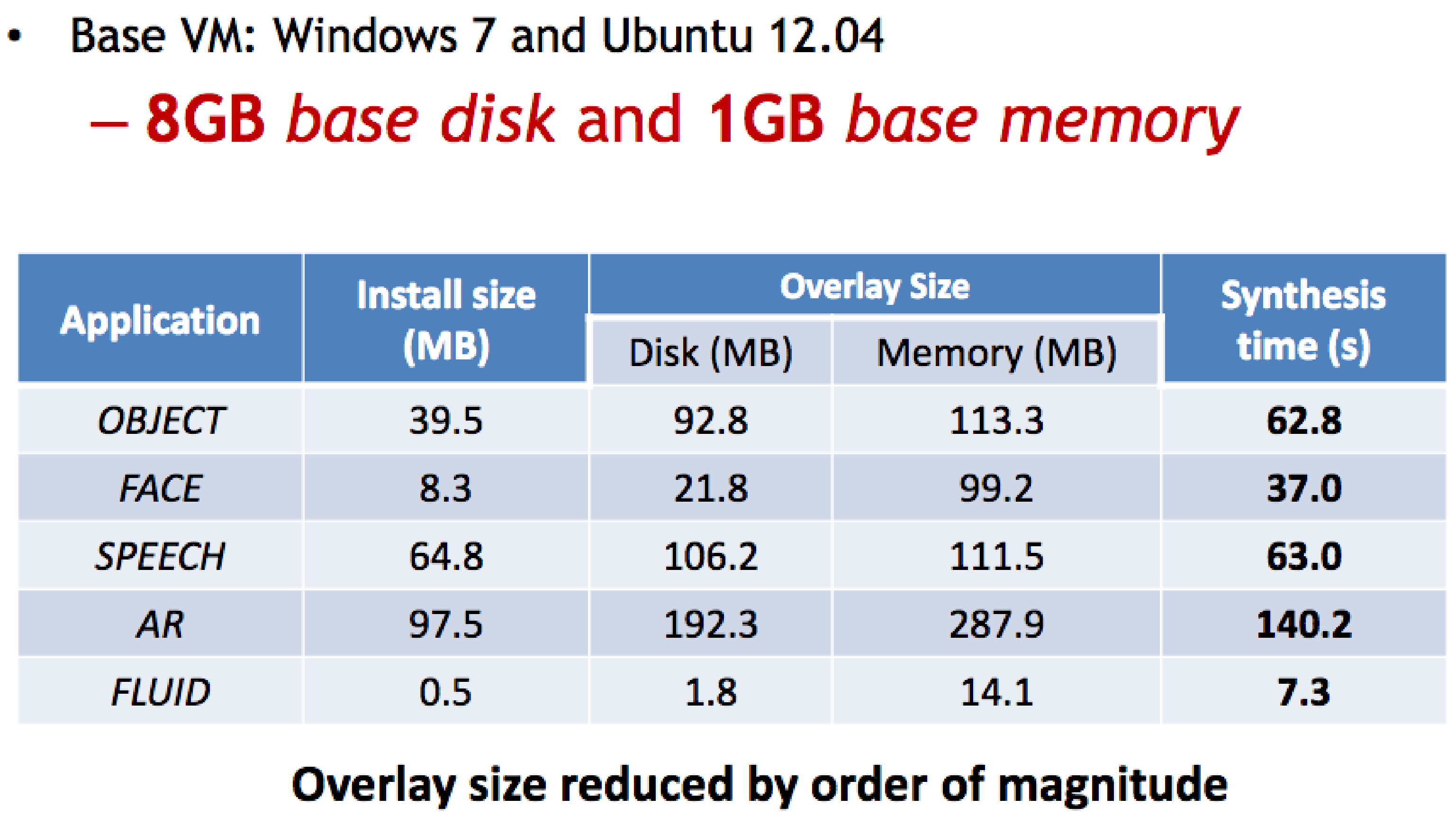
Cloudlet based code offload



Cloudlet based code offload


Cloudlet Client
Face Recognition Client
Cloudlet Server
Cloudlet Host
Guest VM
Cloudlet Startup
CloudletIDPing
Face Recognition Server
Subscribe to File Changes
Start Windows Service
Execute Python Script
Start Server
Face Recognition Server IP Address and Port [HTTP]
Heartbeat [HTTP]
Write Face Recognition Server IP Address/Port to File
Notification of File Changes
Read FRS Address and Port
Heartbeat is sent periodically
Face Recognition Server IP Address/Port [HTTP]
Launch Client
Open Socket (Face Recognition Server IP Address/Port)
Send Image
Image Match Results
Operations repeated until Face Recognition Client is closed

VM Synthesis Baseline Performance



- Base VM: Windows 7 and Ubuntu 12.04
  - 8GB *base disk* and 1GB *base memory*

Application	Install size (MB)	Overlay Size		Synthesis time (s)
		Disk (MB)	Memory (MB)	
OBJECT	39.5	92.8	113.3	**62.8**
FACE	8.3	21.8	99.2	**37.0**
SPEECH	64.8	106.2	111.5	**63.0**
AR	97.5	192.3	287.9	**140.2**
FLUID	0.5	1.8	14.1	**7.3**

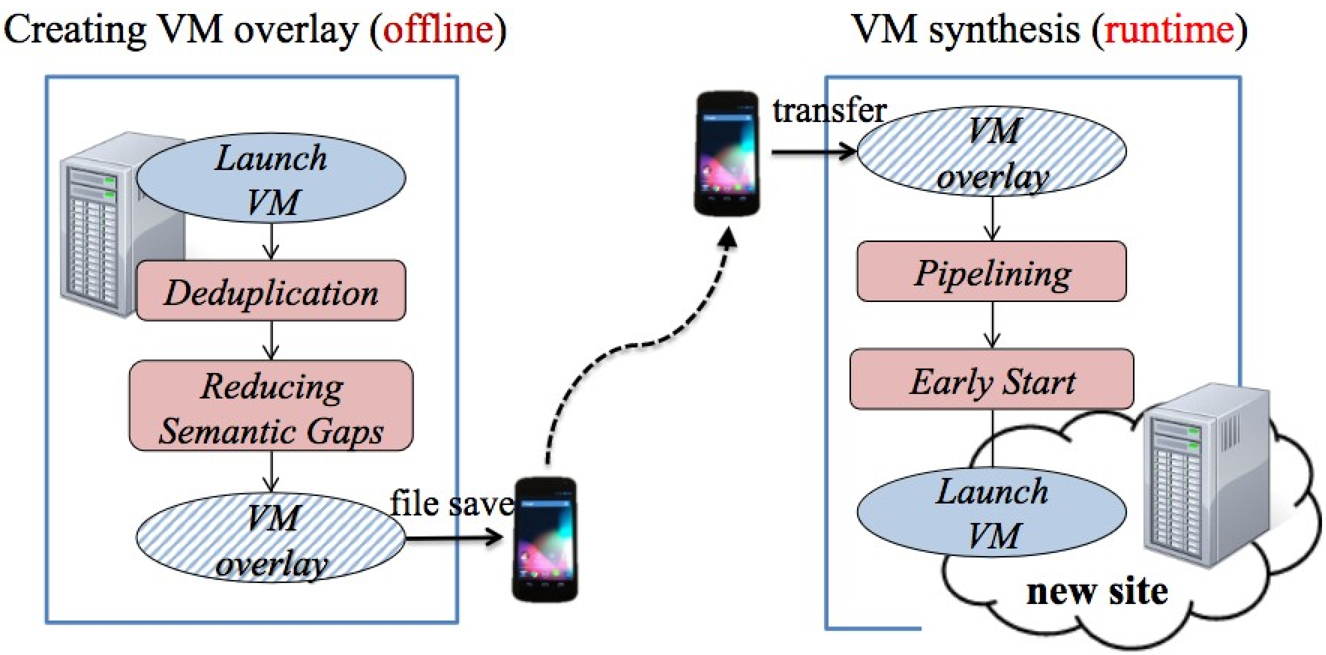
Overlay size reduced by order of magnitude

What does this table tell us?

Overview of Optimizations

1. Minimize VM Overlay

2. Accelerate VM Synthesis



Deduplication

Approach

- Remove redundancy in the VM overlay
  – problem: same bits in base VM and VM overlay but in different locations in the respective images => delta fails

- Sources of redundancy

  Within base VM
  - Shared library copied from base disk
  - Loaded executable binary from base disk

  Between VM overlay's memory and disk
  - Page cache, disk I/O buffer

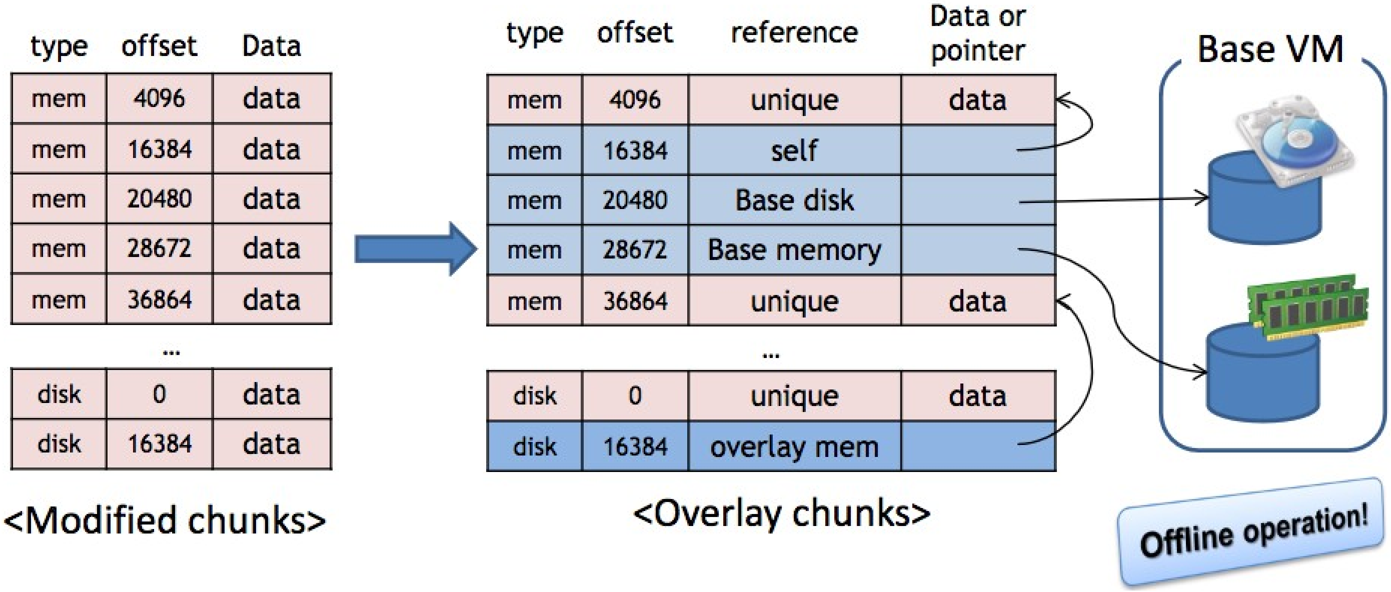
Deduplication

1. Get the list of modified (disk, memory) chunks at the customized VM (delta)

2. Perform deduplication to reduce this list to a minimum

Compare to 1) base disk, 2) base memory, 3) other chunks within itself

Compare between modified memory and modified disk



type	offset	Data
mem	4096	data
mem	16384	data
mem	20480	data
mem	28672	data
mem	36864	data
	...	
disk	0	data
disk	16384	data

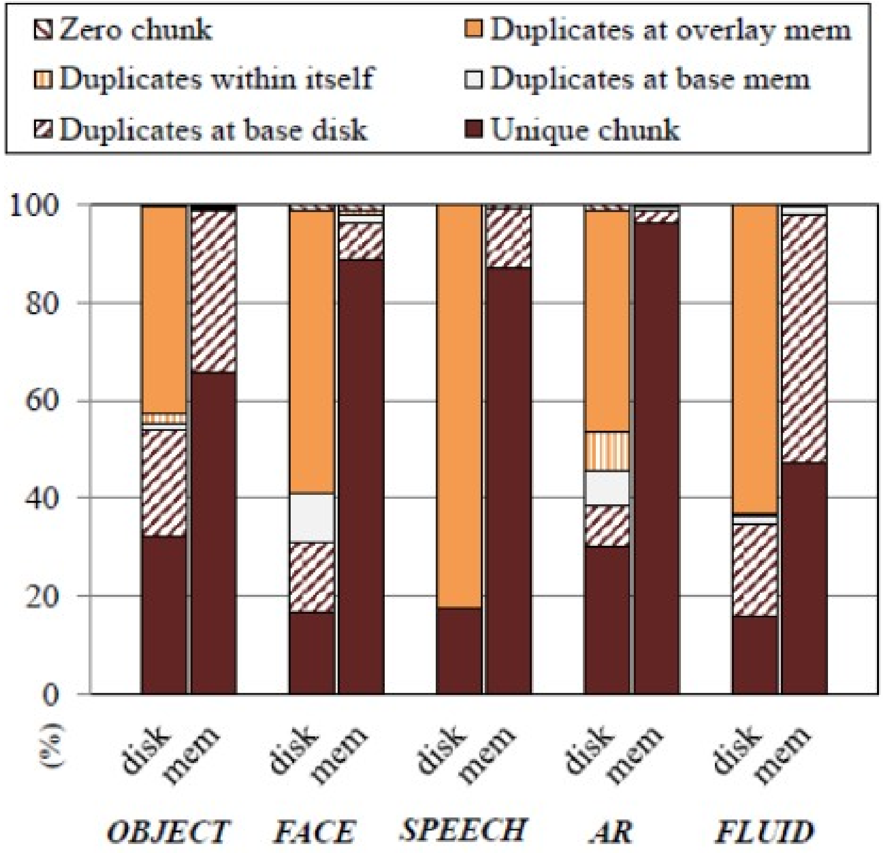
<Modified chunks>

type	offset	reference	Data or pointer
mem	4096	unique	data
mem	16384	self	
mem	20480	Base disk	
mem	28672	Base memory	
mem	36864	unique	data
		...	
disk	0	unique	data
disk	16384	overlay mem	

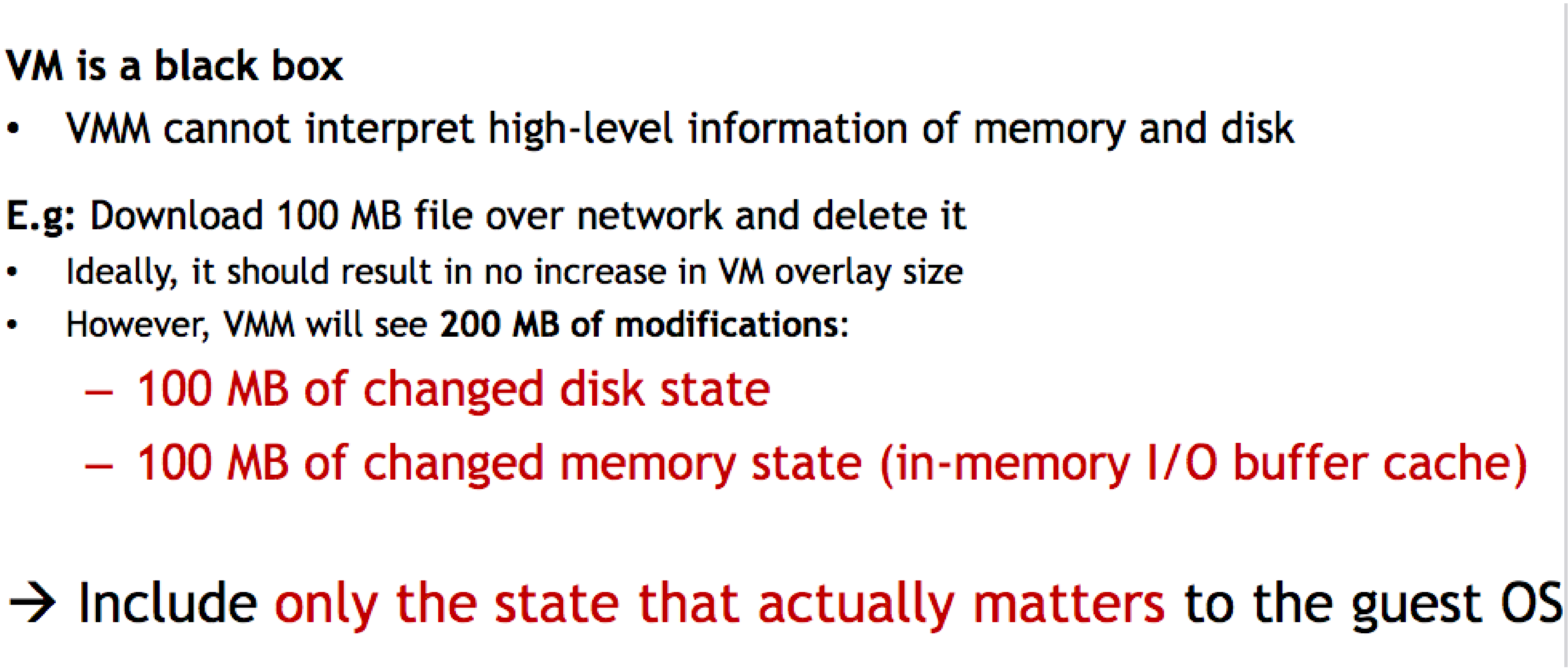
<Overlay chunks>

Offline operation!

Dedup Results



Reduce Semantic Gap



VM is a black box

- VMM cannot interpret high-level information of memory and disk

E.g: Download 100 MB file over network and delete it

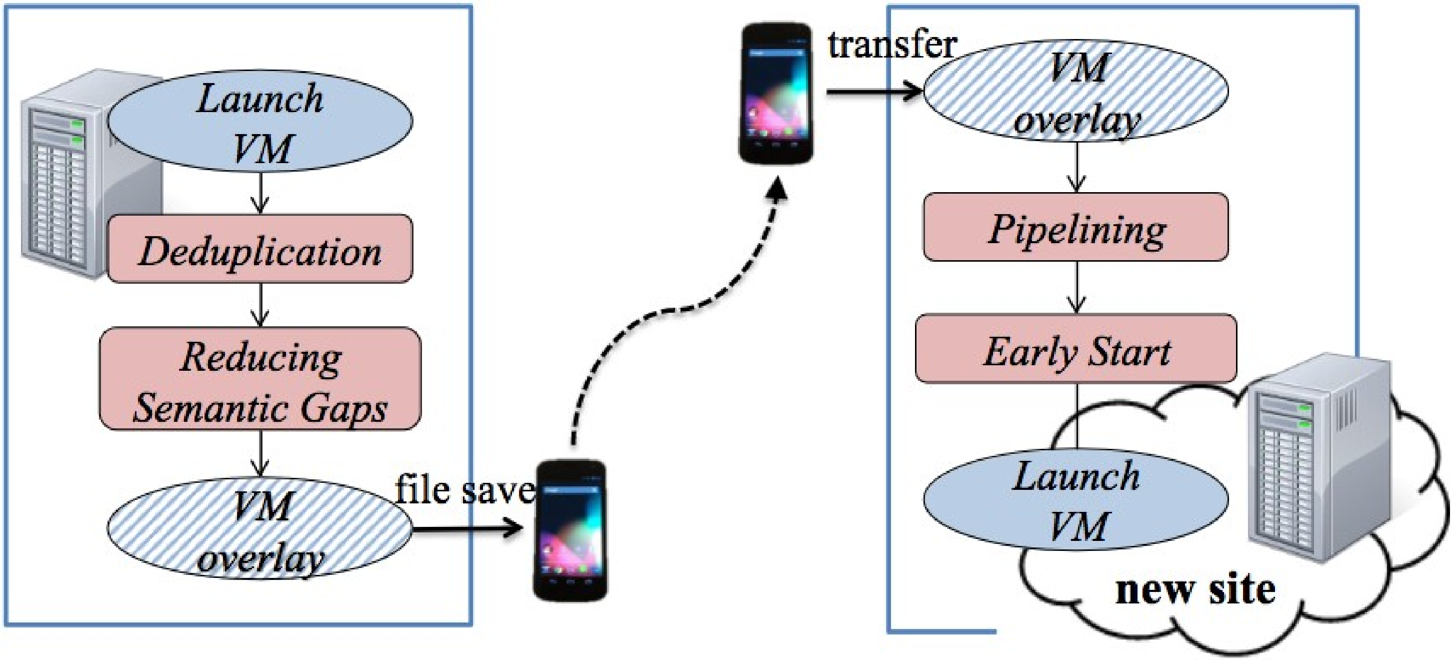
- Ideally, it should result in no increase in VM overlay size
- However, VMM will see **200 MB of modifications:**
  - 100 MB of changed disk state
  - 100 MB of changed memory state (in-memory I/O buffer cache)

→ Include only the state that actually matters to the guest OS

Overview of Optimization

1. Minimize *VM overlay* size ✓ Creating VM overlay (offline)
2. Accelerate VM synthesis

VM synthesis (runtime)

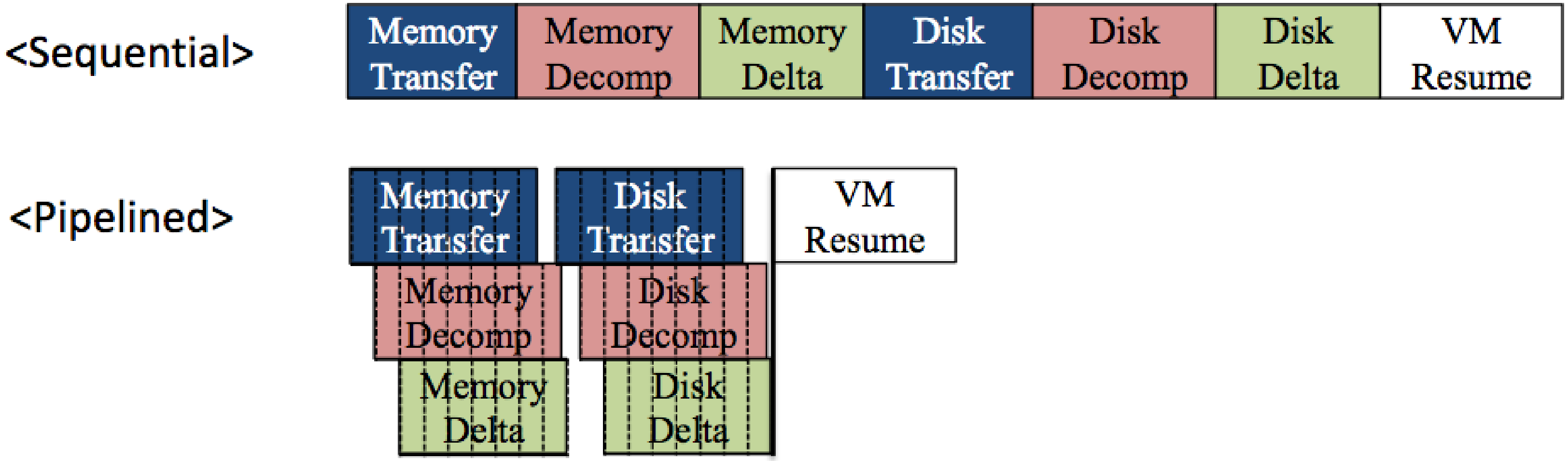


VM synthesis time is still too large

Pipelining

- Steps for VM synthesis

① Transfer VM overlay ② Decompress ③ Apply delta



- Unit of transfer: segment. How big?
  - Bigger more efficient; finer better on latency

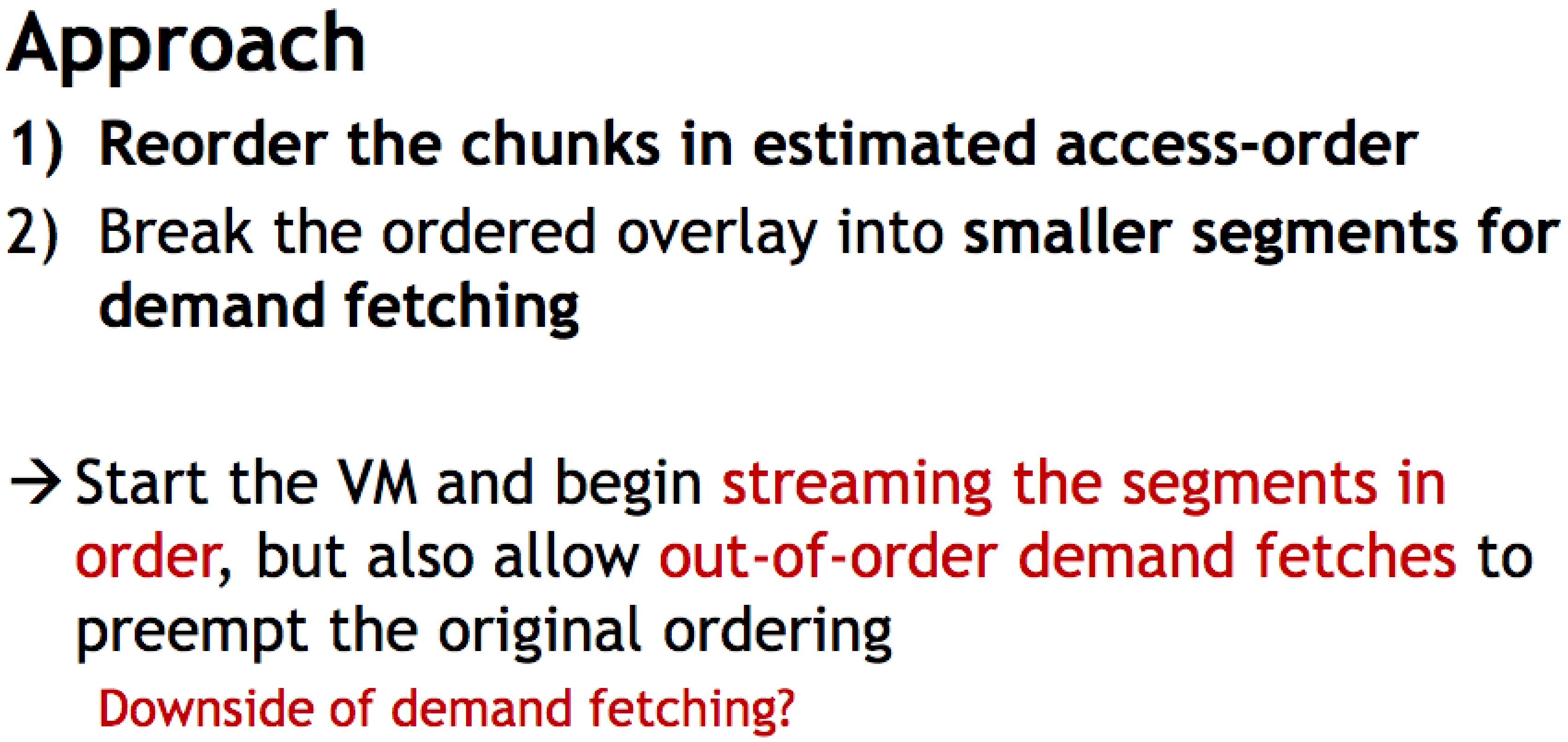
Early Start

Idea

- From user’s perspective, first response time of offloading is most important
- Starting VM even before finishing VM synthesis

→Do not wait until VM synthesis finishes, but start offloading immediately and process the request while synthesis is ongoing

Early Start



Approach

1) **Reorder the chunks in estimated access-order**
2) Break the ordered overlay into **smaller segments for demand fetching**

→ Start the VM and begin streaming the segments in order, but also allow out-of-order demand fetches to preempt the original ordering

Downside of demand fetching?