SJTU SAIL (Software Architecture and Infrastructure Lab)

SHANGHAI JIAO TONG UNIVERSITY

Serverless

Yalun Lin Hsu & Yuchen Cheng

2020-09-24

In 2009 ……

- The appearance of infinite computing resources on demand.
- The elimination of an up-front commitment by cloud users.
- The ability to pay for use of computing resources on a short-term basis as needed.
- Economies of scale that significantly reduced cost due to many, very large data centers.
- Simplifying operation and increasing utilization via resource virtualization.
- Higher hardware utilization by multiplexing workloads from different organizations.

Status Quo

- The appearance of infinite computing resources on demand.
- The elimination of an up-front commitment by cloud users.
- The ability to pay for use of computing resources on a short-term basis as needed.
- Economies of scale that significantly reduced cost due to many, very large data centers.

- Simplifying operation and increasing utilization via resource virtualization.
- Higher hardware utilization by multiplexing workloads from different organizations.

EC2

Downside of EC2

- Redundancy for availability, so that a single machine failure doesn't take down the service.
- Geographic distribution of redundant copies to preserve the service in case of disaster.
- Load balancing and request routing to efficiently utilize resources.
- Autoscaling in response to changes in load to scale up or down the system.
- Monitoring to make sure the service is still running well.
- Logging to record messages needed for debugging or performance tuning.
- System upgrades, including security patching.
- Migration to new instances as they become available.

AWS Lambda

Serverless Computing = FaaS + BaaS

Introducing Serverless

	Characteristic	*AWS Serverless Cloud*	*AWS Serverful Cloud*
PROGRAMMER	When the program is run	On event selected by Cloud user	Continuously until explicitly stopped
	Programming Language	JavaScript, Python, Java, Go, C#, etc.[4]	Any
	Program State	Kept in storage (stateless)	Anywhere (stateful or stateless)
	Maximum Memory Size	0.125 - 3 GiB (Cloud user selects)	0.5 - 1952 GiB (Cloud user selects)
	Maximum Local Storage	0.5 GiB	0 - 3600 GiB (Cloud user selects)
	Maximum Run Time	900 seconds	None
	Minimum Accounting Unit	0.1 seconds	60 seconds
	Price per Accounting Unit	$0.0000002 (assuming 0.125 GiB)	$0.0000867 - $0.4080000
	Operating System & Libraries	Cloud provider selects[5]	Cloud user selects
SYSADMIN	Server Instance	Cloud provider selects	Cloud user selects
	Scaling[6]	Cloud provider responsible	Cloud user responsible
	Deployment	Cloud provider responsible	Cloud user responsible
	Fault Tolerance	Cloud provider responsible	Cloud user responsible
	Monitoring	Cloud provider responsible	Cloud user responsible
	Logging	Cloud provider responsible	Cloud user responsible

Serverless vs Serverful

- Decoupled computation and storage.
- Executing code without managing resource allocation.
- Paying in proportion to resources used instead of for resources allocated.

Attractiveness of Serverless

Percent	*Use Case*
32%	Web and API serving
21%	Data Processing, e.g., batch ETL (database Extract, Transform, and Load)
17%	Integrating 3rd Party Services
16%	Internal tooling
8%	Chat bots e.g., Alexa Skills (SDK for Alexa AI Assistant)
6%	Internet of Things

Limitations

Inadequate storage for fine-grained operations

		Block Storage (e.g., AWS EBS, IBM Block Storage)	Object Storage (e.g., AWS S3, Azure Blob Store, Google Cloud Storage)	File System (e.g., AWS EFS, Google Filestore)	Elastic Database (e.g., Google Cloud Datastore, Azure Cosmos DB)	Memory Store (e.g., AWS ElastiCache, Google Cloud Memorystore)	**"Ideal" storage service for serverless computing**
Cloud functions access		No	Yes	Yes[13]	Yes	Yes	Yes
Transparent Provisioning		No	Yes	Capacity only[14]	Yes[15]	No	Yes
Availability and persistence guarantees		Local & Persistent	Distributed & Persistent	Distributed & Persistent	Distributed & Persistent	Local & Ephemeral	Various
Latency (mean)		< 1ms	$10 - 20$ms	$4 - 10$ms	$8 - 15$ms	< 1ms	< 1ms
Cost[16]	Storage capacity (1 GB/month)	$0.10	$0.023	$0.30	$0.18–$0.25	$1.87	∼$0.10
	Throughput (1 MB/s for 1 month)	$0.03	$0.0071	$6.00	$3.15-$255.1	$0.96	∼$0.03
	IOPS (1/s for 1 month)	$0.03	$7.1	$0.23	$1-$3.15	$0.037	∼$0.03

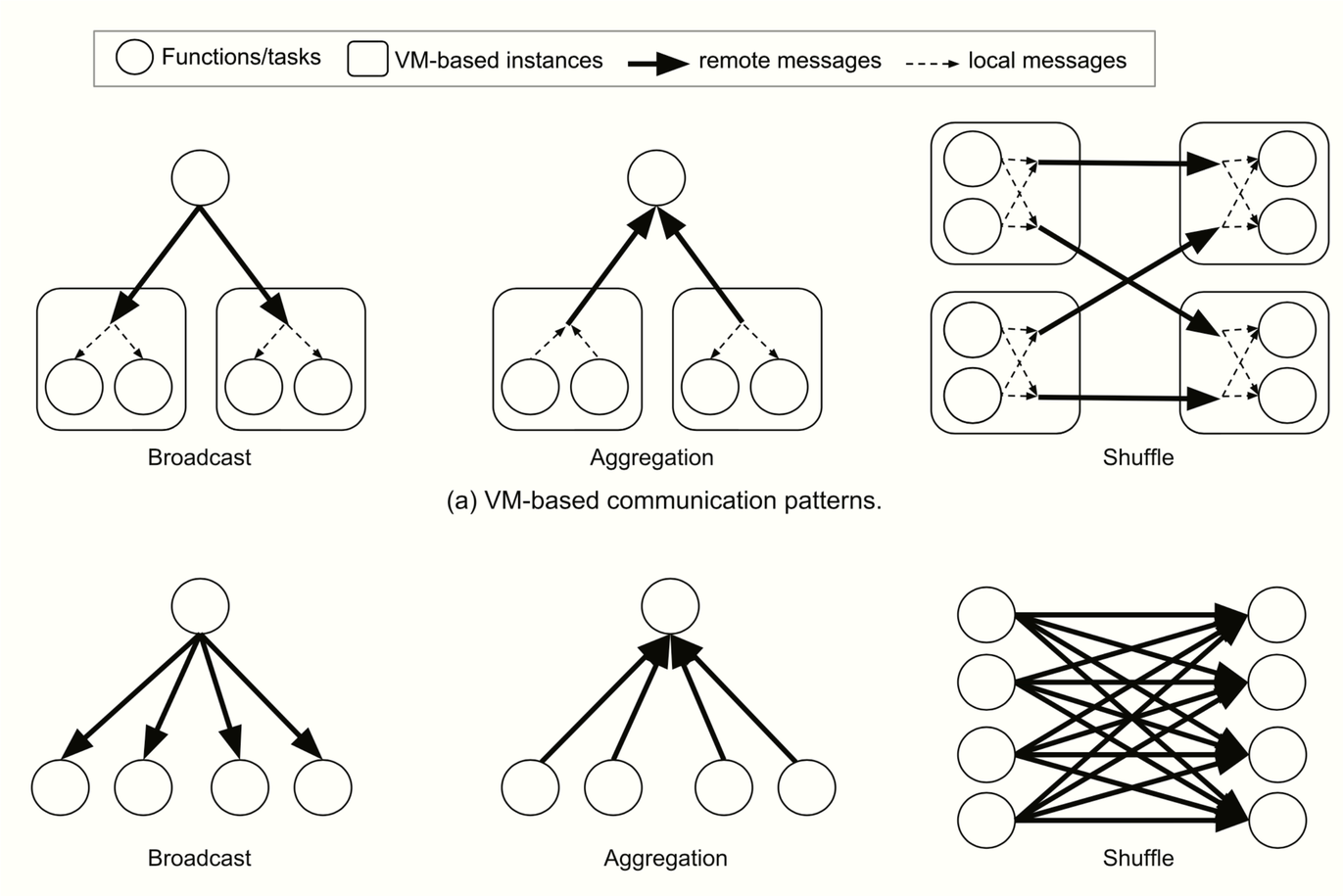
Limitations

Lack of fine-grained coordination

Task A uses task B’s output there must be a way for A to know when its input is available, even if A and B reside on different nodes.

Limitations

Poor performance for standard communication patterns



(a) VM-based communication patterns.

Limitations

Predictable Performance

- The time it takes to start a cloud function
- The time it takes to initialize the software environment of the function
- Application-specific initialization in user code

Challenges

System Challenges

- **High-performance, affordable, transparently provisioned storage;**
- **Coordination/signaling service;**
- **Minimize startup time.**

Challenges

Abstraction Challenges

- **Resource Requirement**
  - Serverless hinders those who want more control on specifying resources;
  - Having the cloud provider infer resource requirements instead of having the developer specify them;
  - Provisioning just the right amount of memory automatically is particularly appealing.

Challenges

Abstraction Challenges

- **Data dependencies**
  - Platforms have no knowledge of the data dependencies between the cloud functions, let alone the amount of data these functions might exchange;
  - Using computation graph API;
  - Modify MapReduce, Spark, Beam, Cloud Dataflow, Airflow …

USENIX ATC '18

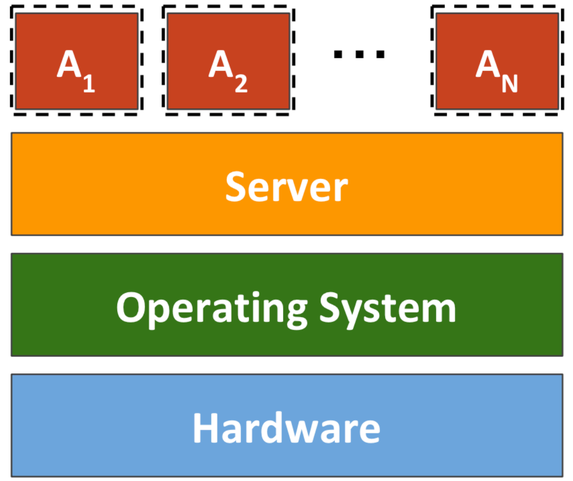
SOCK

Rapid Task Provisioning with Serverless-Optimized Containers

https://www.usenix.org/conference/atc18/presentation/oakes

Serverless Runtime

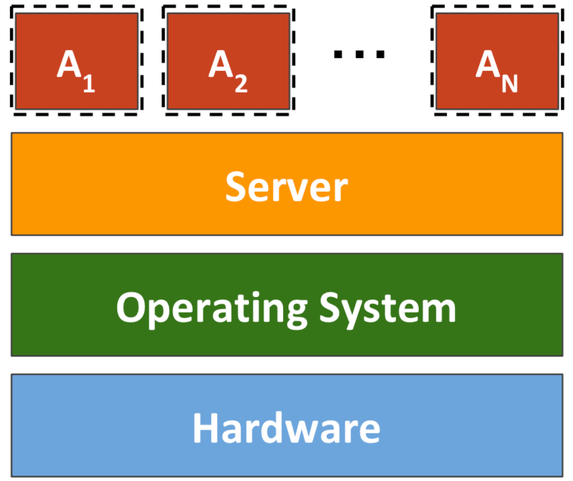
Docker Container: 400 ms



Serverless Runtime

Docker Container: 400 ms

Python Interpreter: 30 ms



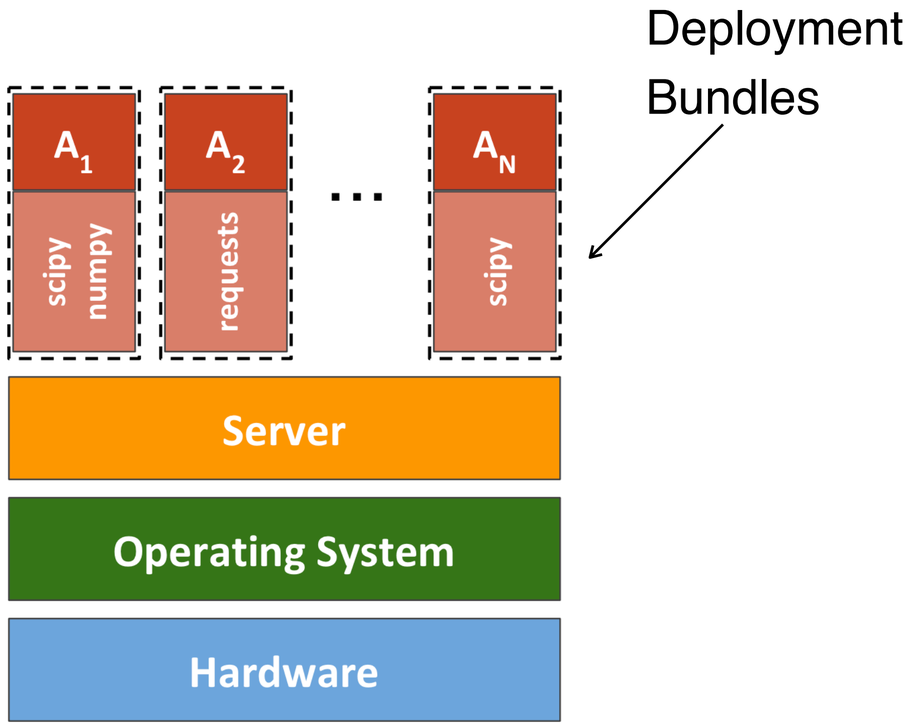
Serverless Runtime

Docker Container: 400 ms

Python Interpreter: 30 ms

scipy:

* 2700 ms download
* 8200 ms install
* 88 ms import



SOCK

- **Lean serverless-optimized containers (SOCK)**
  - Precise usage of Linux isolation mechanisms
  - 18x faster container lifecycle over Docker
- **Provision from secure Zygote processes**
  - Fork from initialized runtime to prevent cold start
  - 3x faster provisioning than SOCK alone
- **Execution caching across 3 tiers**
  - Securely reuse initialization work across customers
  - 3-16x lower platform cost in image-processing case study

SOCK

- **Lean serverless-optimized containers (SOCK)**
  - Precise usage of Linux isolation mechanisms
  - 18x faster container lifecycle over Docker

- **Provision from secure Zygote processes**
  - Fork from initialized runtime to prevent cold start
  - 3x faster provisioning than SOCK alone

- **Execution caching across 3 tiers**
  - Securely reuse initialization work across customers
  - 3-16x lower platform cost in image-processing case study

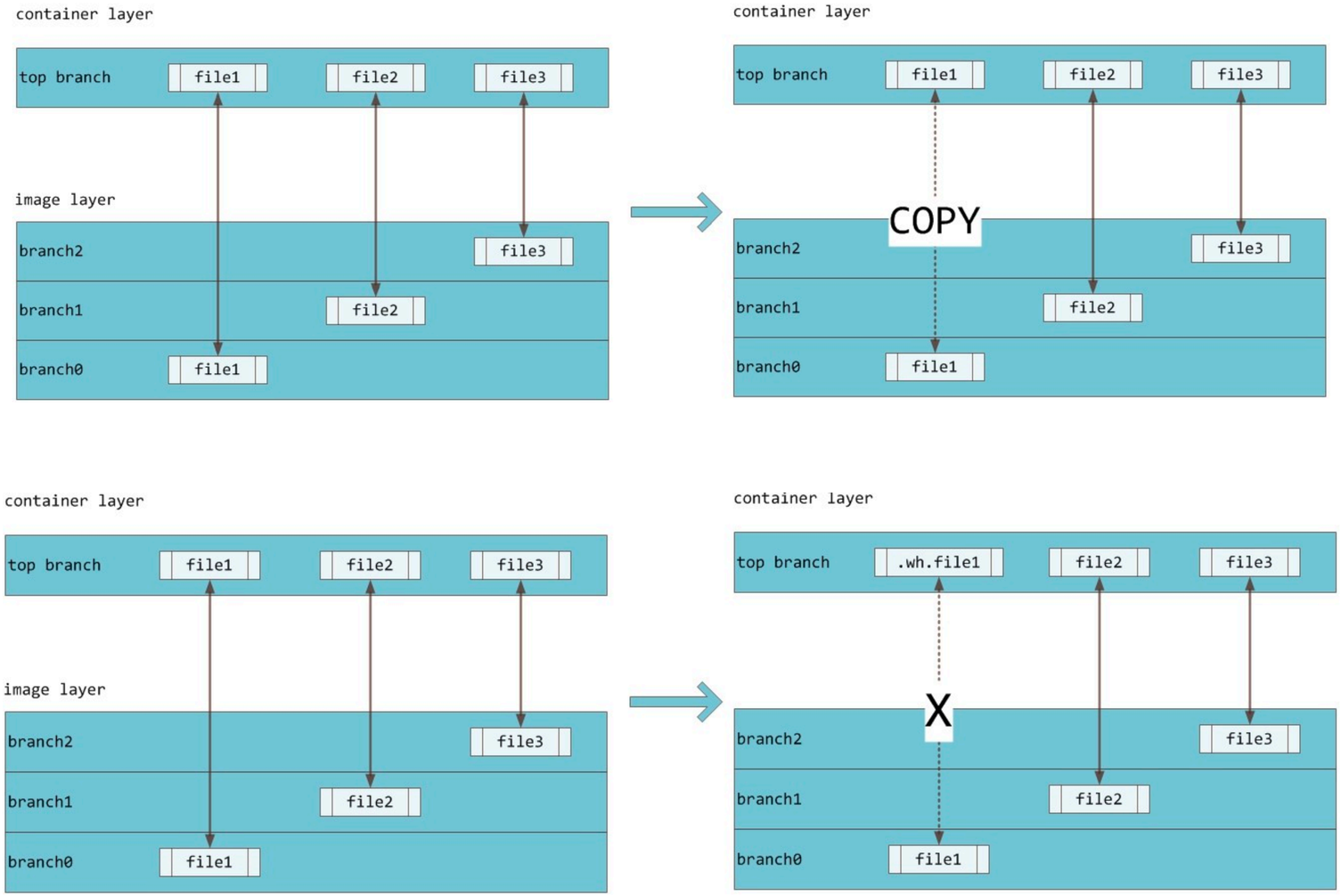
Linux Containers

…they're just cheaper VMs, right?

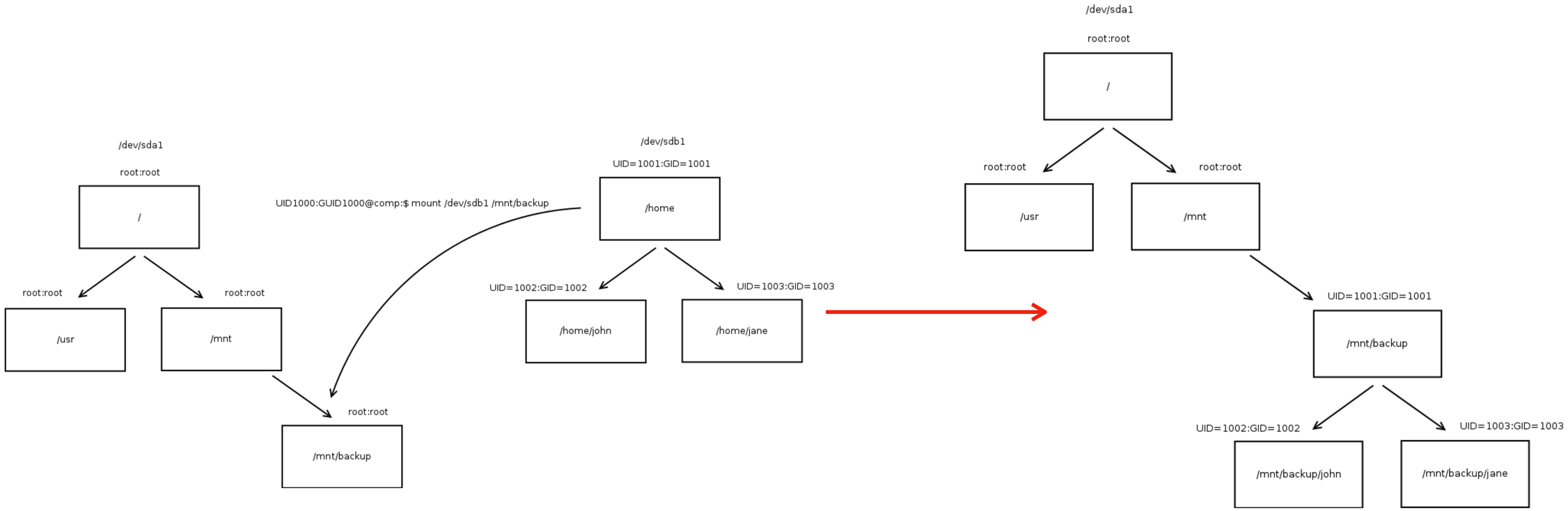
- How can one maximize density of container file systems per machine?
- What is the cost of isolating each major resource with namespaces, and which resources must be isolated in a serverless environment?
- And how can the cost of repeatedly initializing cgroups to isolate performance be avoided?

Container Storage

AUFS

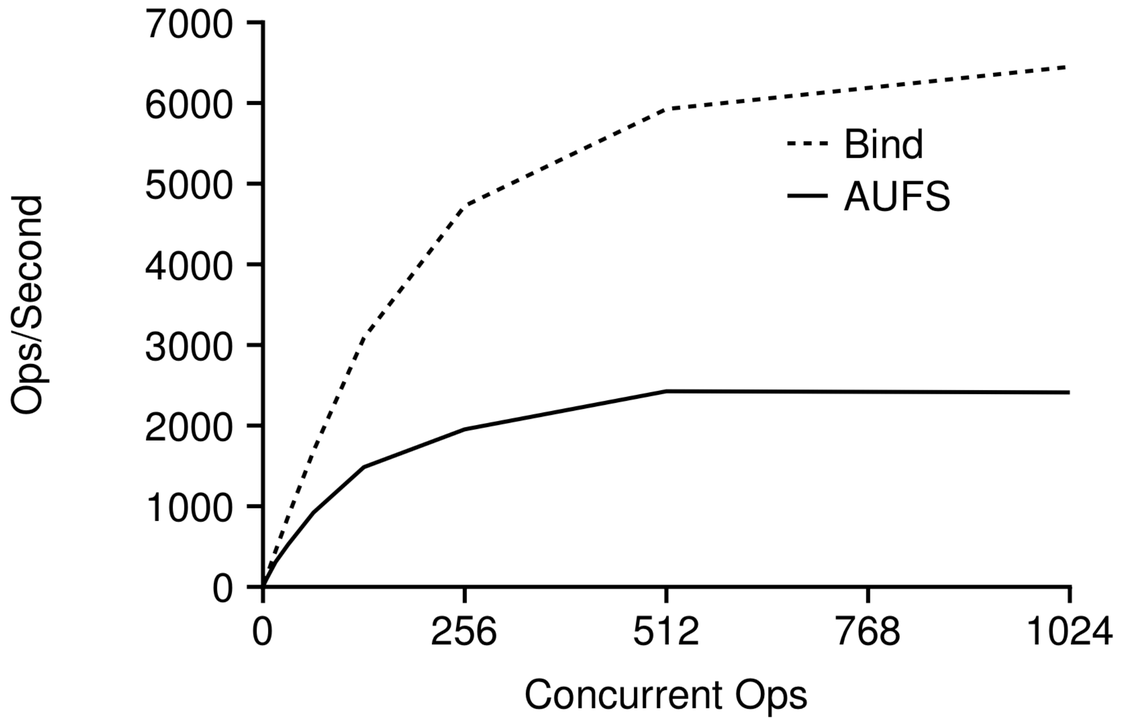


Container Storage

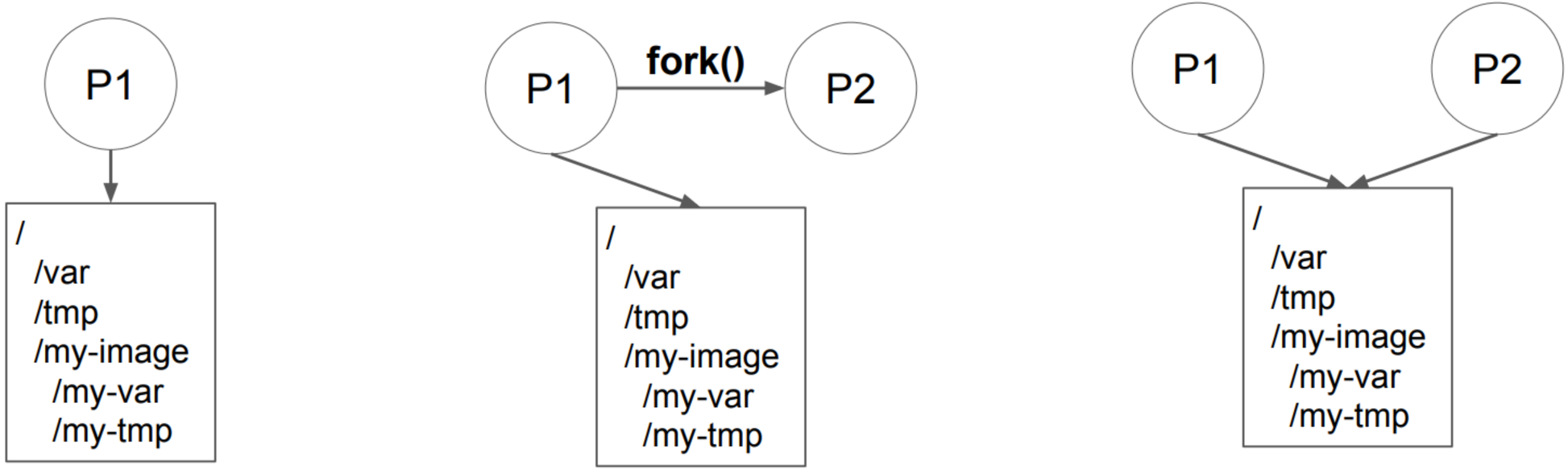


mount -o bind

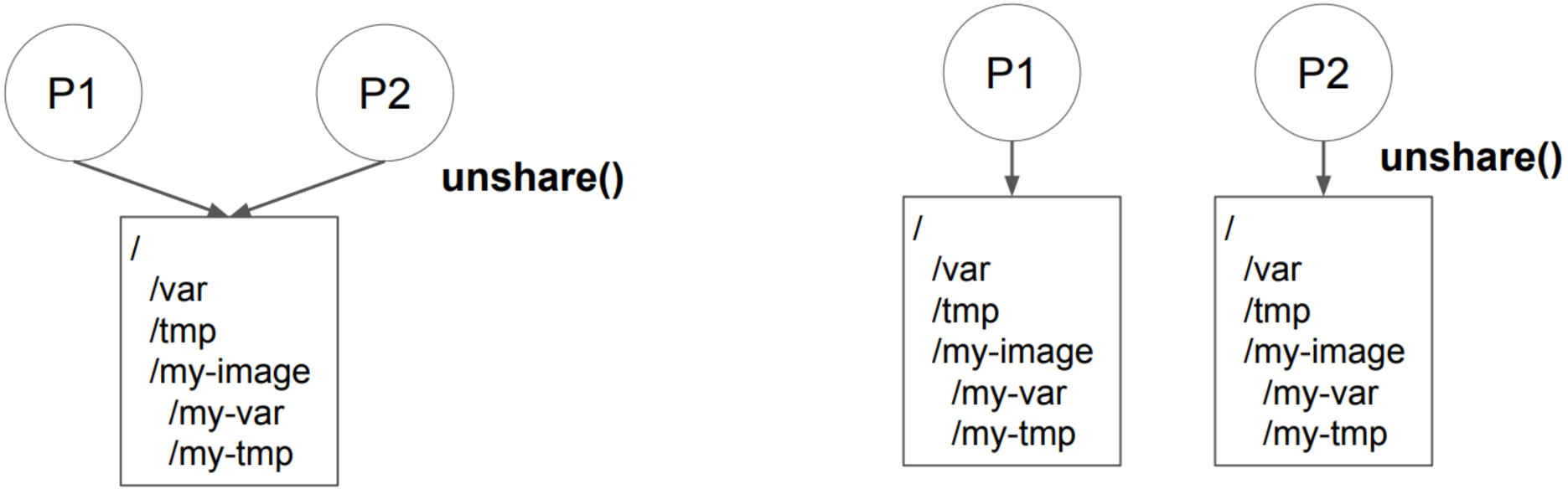
Mount Performance



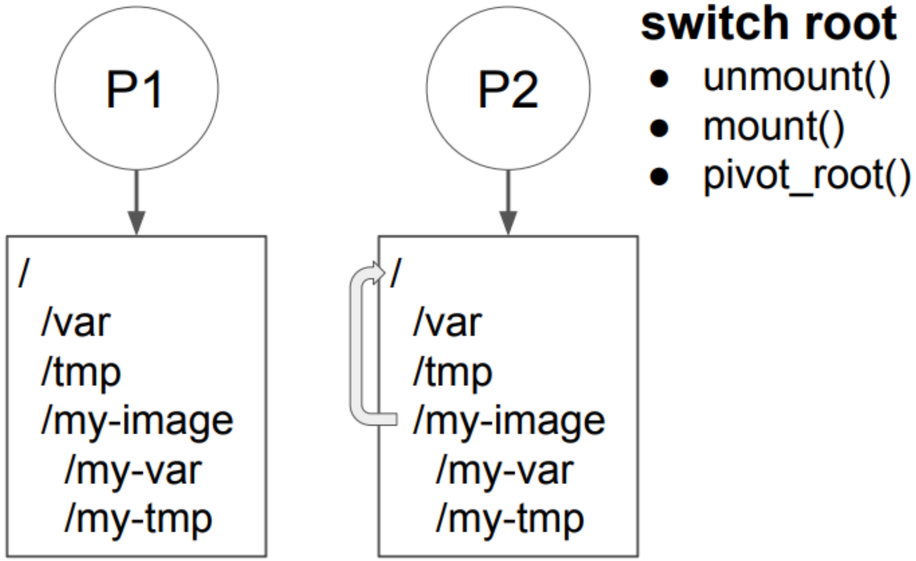
Mount Namespace

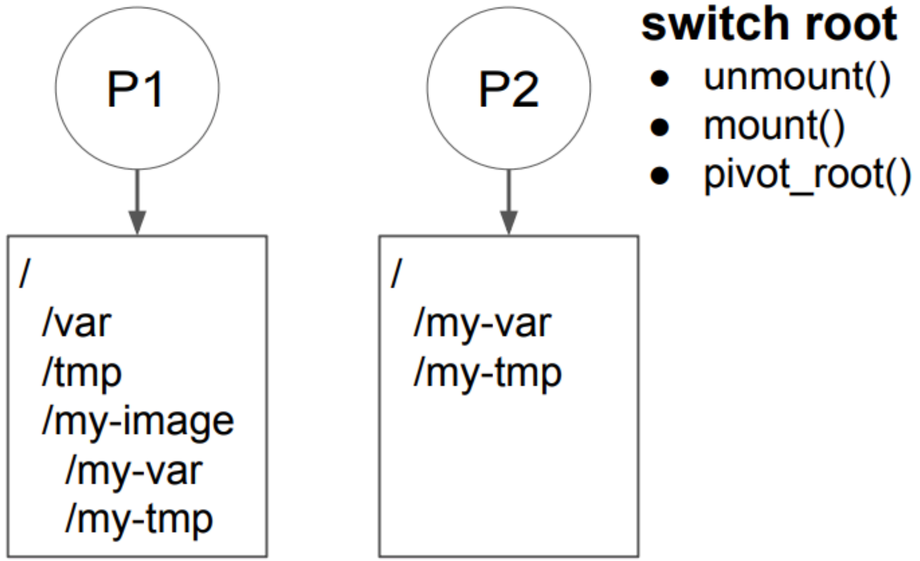


Mount Namespace

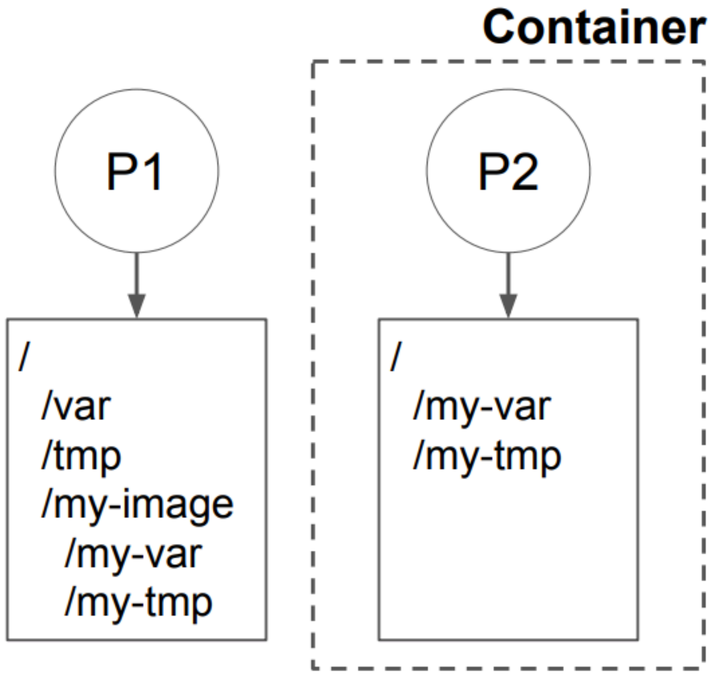


Mount Namespace

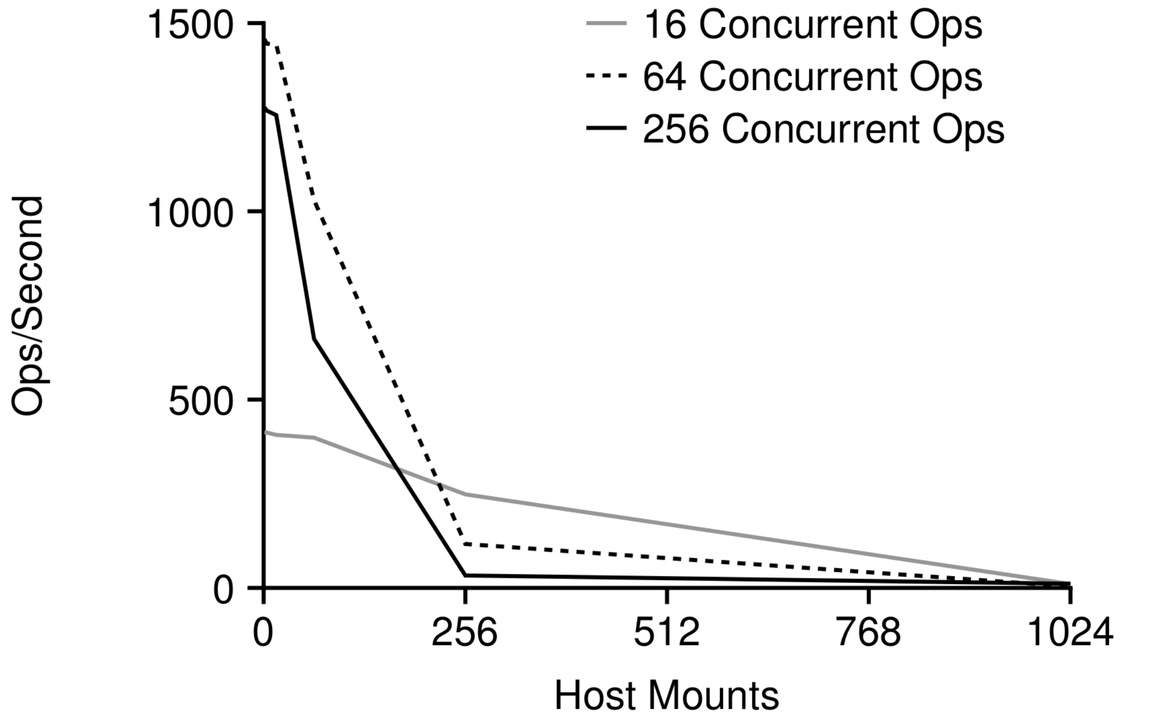




Mount Namespace



Mount Scalability

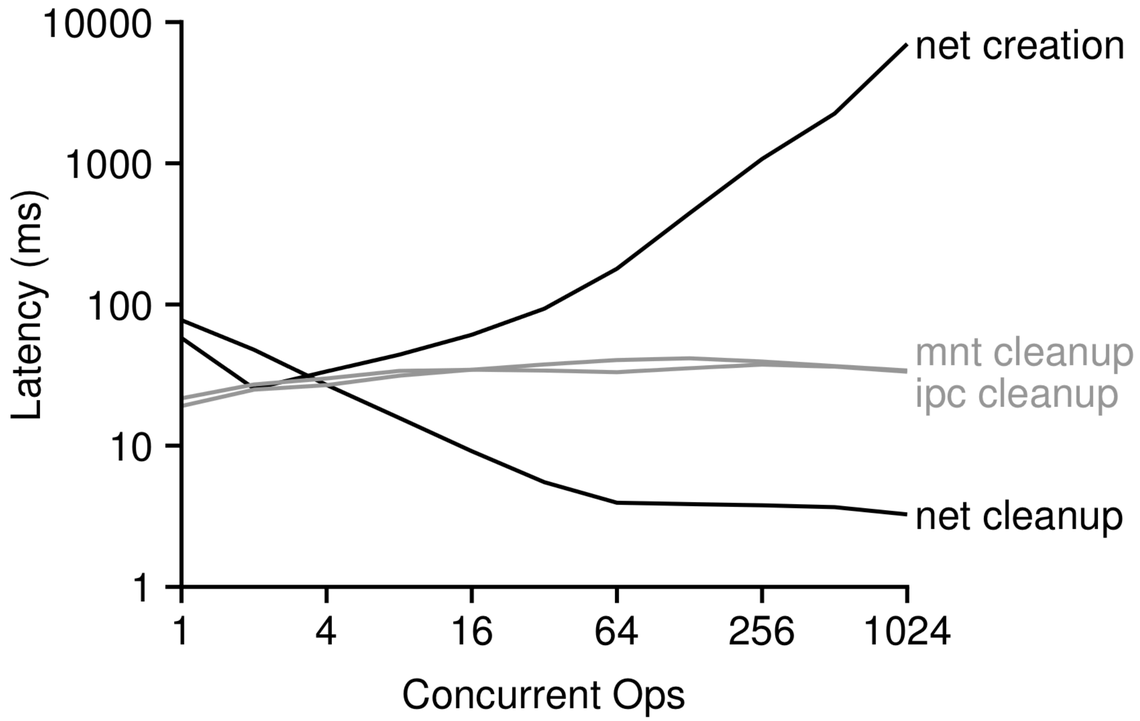


Namespaces

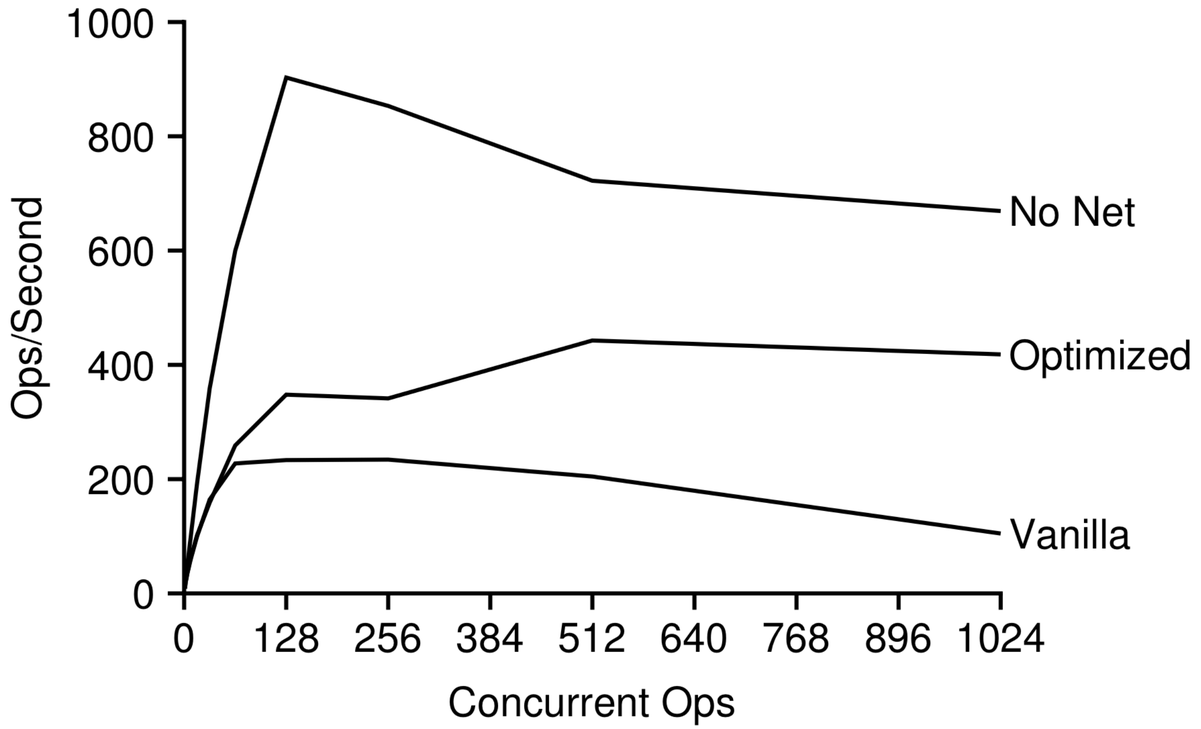
- Partition resource access in the kernel

- 7 individual namespaces
  - Mount
  - Network
  - User
  - UTS
  - IPC
  - PID
  - Cgroup

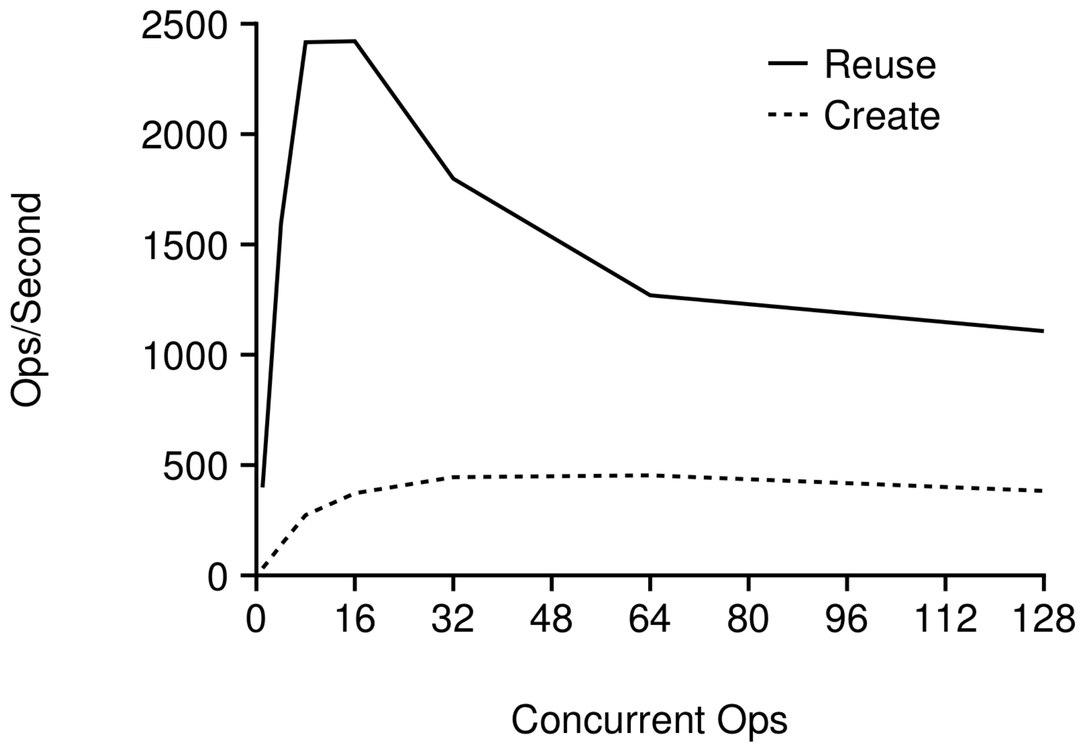
Namespace Primitives



Network Namespace



Cgroup Primitives

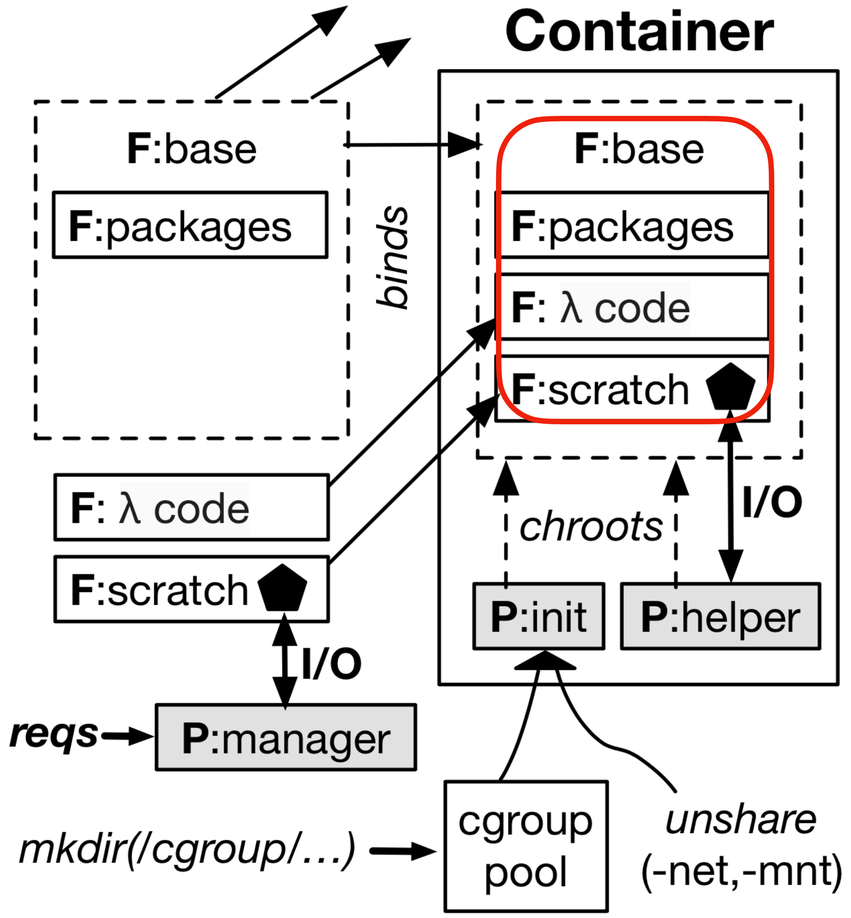


Serverless Impaction

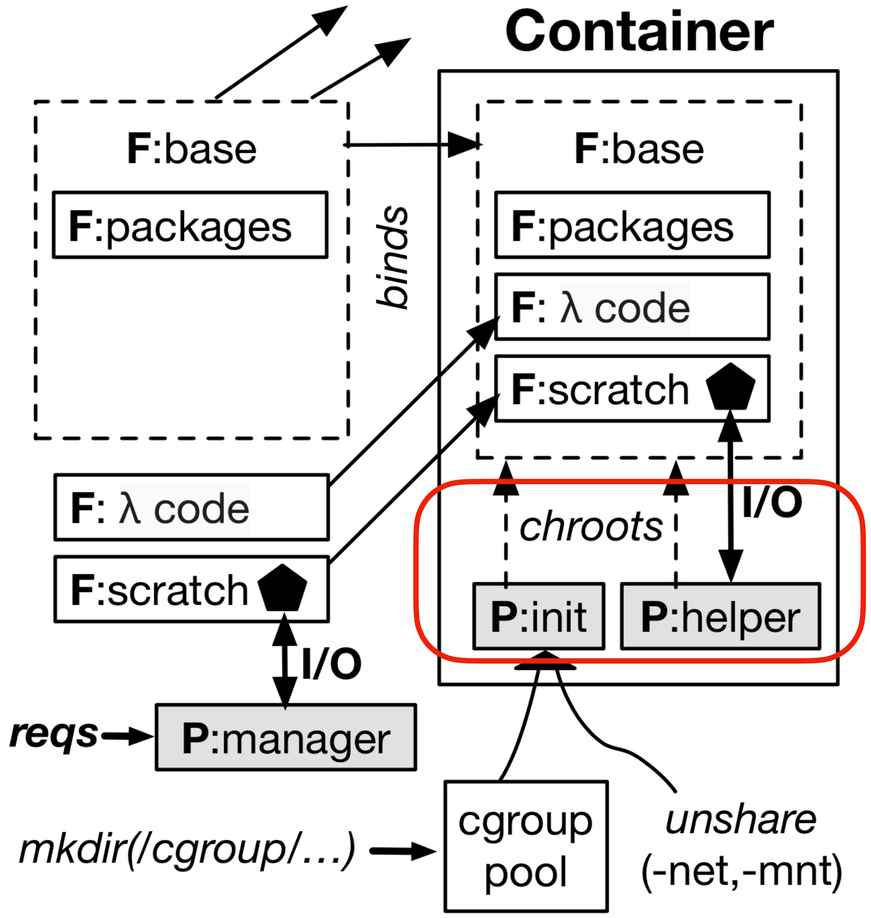
Replace flexible, costly mechanisms with simple, cheap alternatives.

- AUFS + mount NS -> bind mounts + chroot
- network NS -> domain socket + outbound access
- Cgroup create/delete -> Cgroup reuse

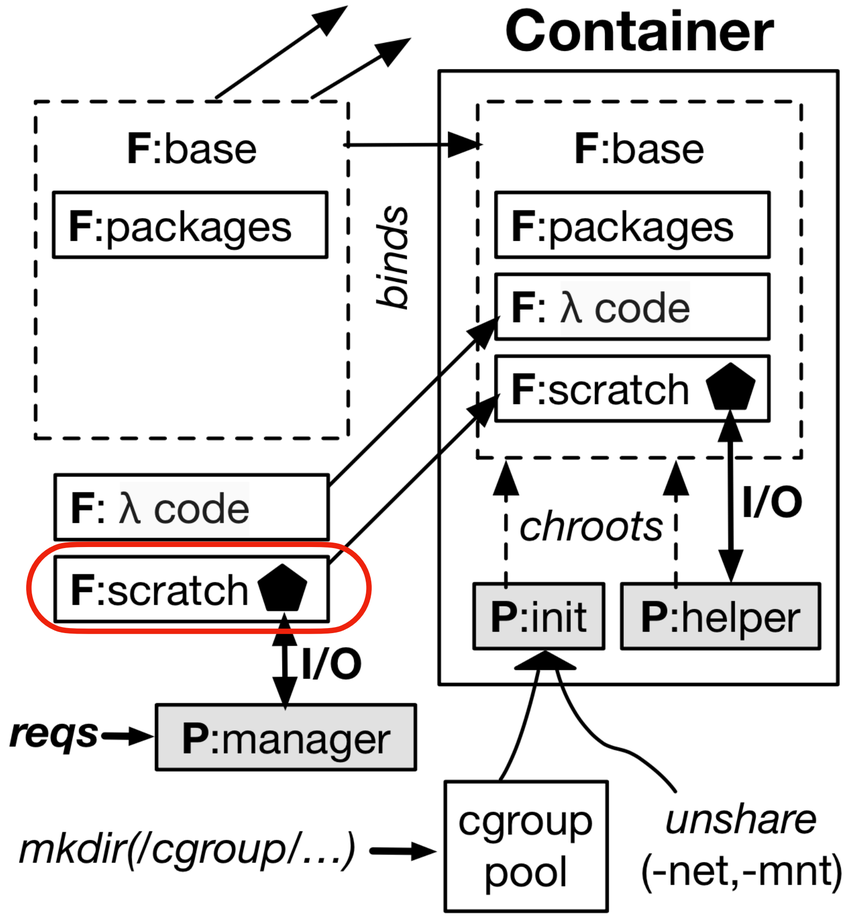
Lean Container



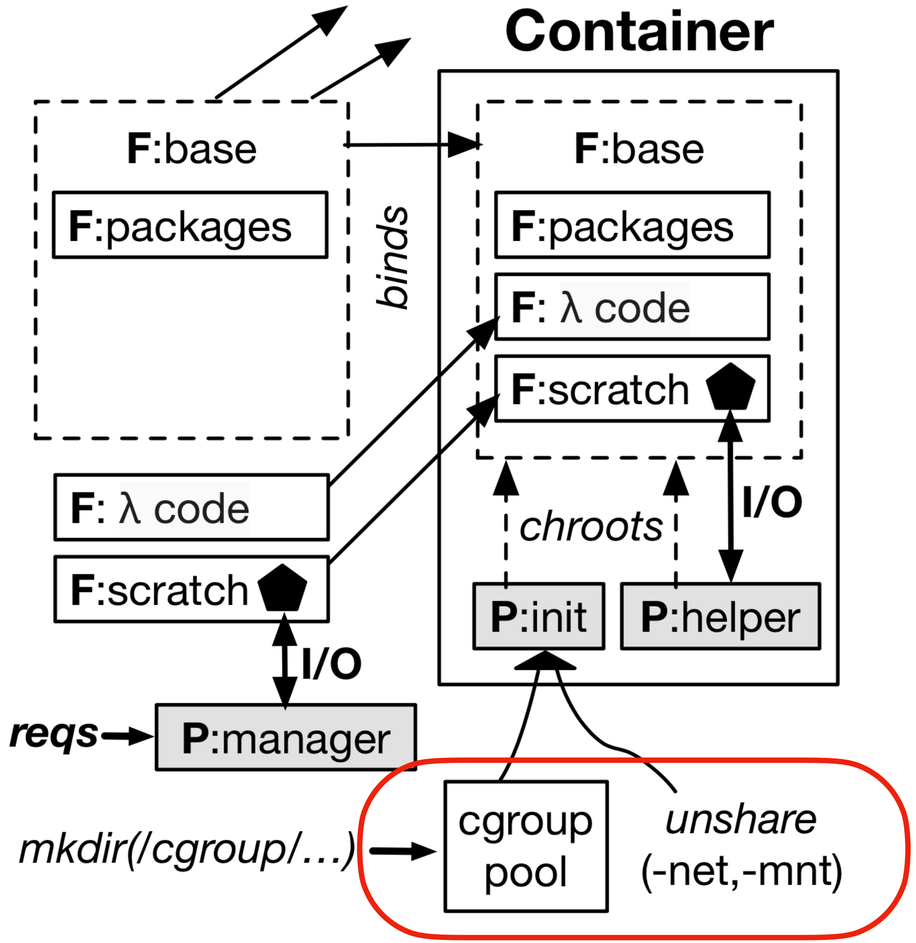
Lean Container



Lean Container



Lean Container



SOCK

- **Lean serverless-optimized containers (SOCK)**
  - Precise usage of Linux isolation mechanisms
  - 18x faster container lifecycle over Docker

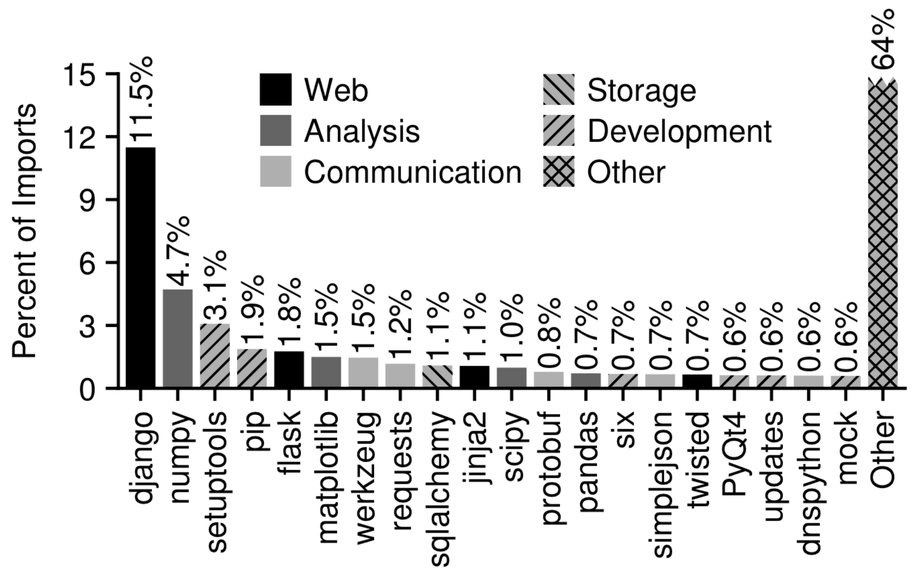
- **Provision from secure Zygote processes**
  - Fork from initialized runtime to prevent cold start
  - 3x faster provisioning than SOCK alone

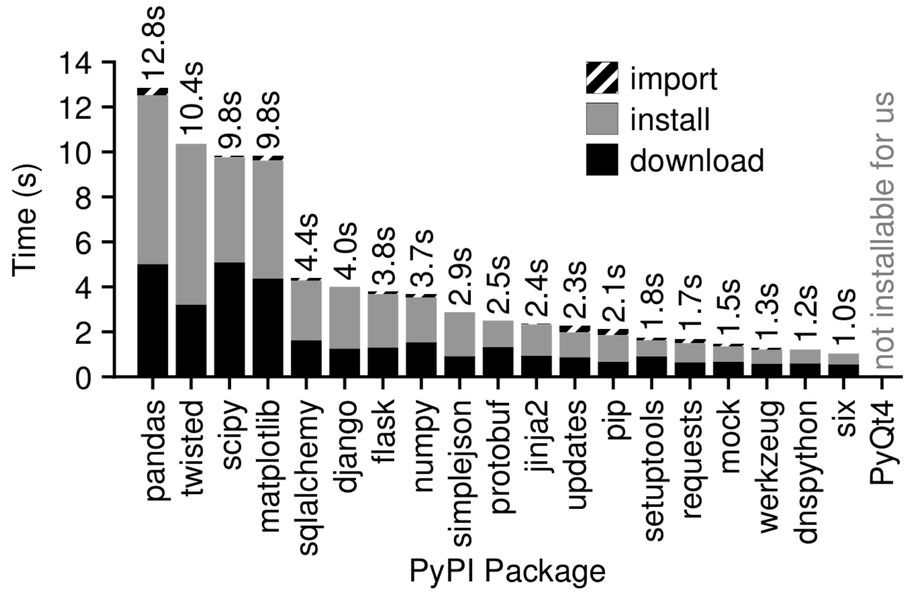
- **Execution caching across 3 tiers**
  - Securely reuse initialization work across customers
  - 3-16x lower platform cost in image-processing case study

Python Applications

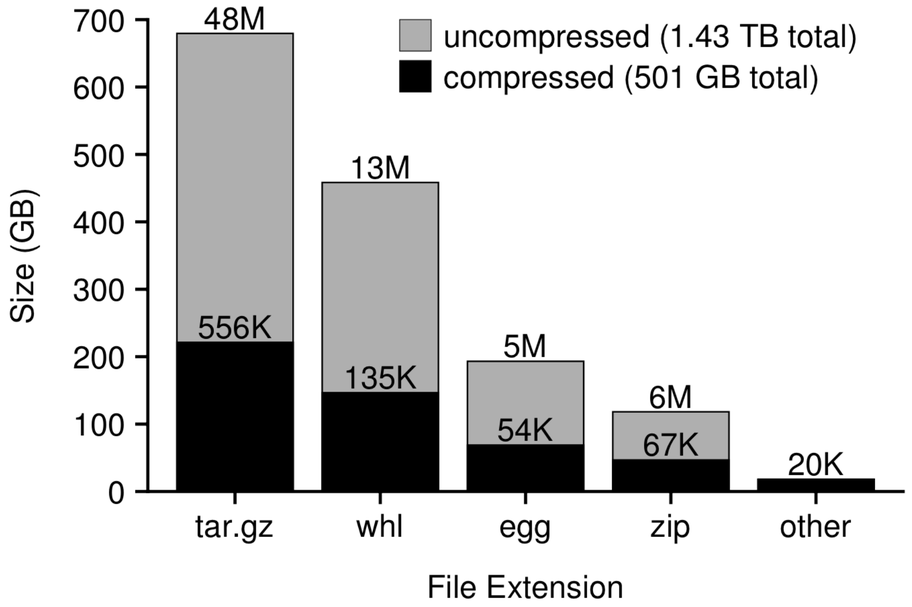
- What types of packages are most popular in Python applications?
- What are the initialization costs associated with using these packages?
- How feasible is it to cache a large portion of mainstream package repositories on local lambda workers?

Python Applications





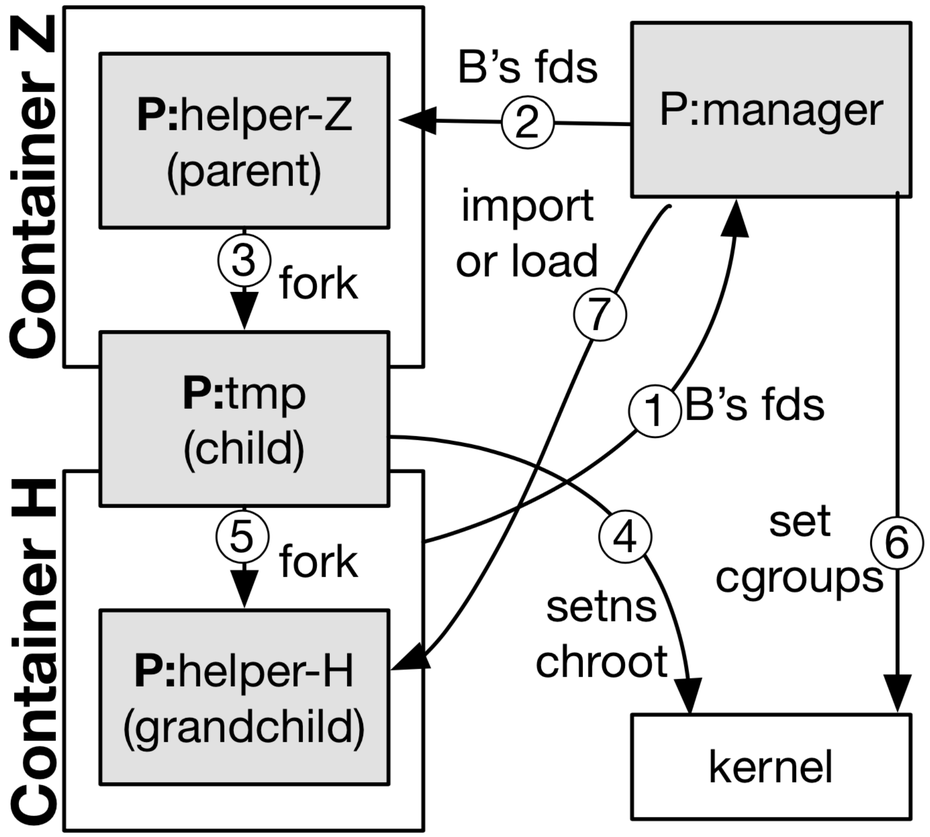
PyPi Repo


uncompressed (1.43 TB total)
compressed (501 GB total)
Size (GB)
700
600
500
400
300
200
100
0
48M
556K
13M
135K
5M
54K
6M
67K
20K
tar.gz
whl
egg
zip
other
File Extension

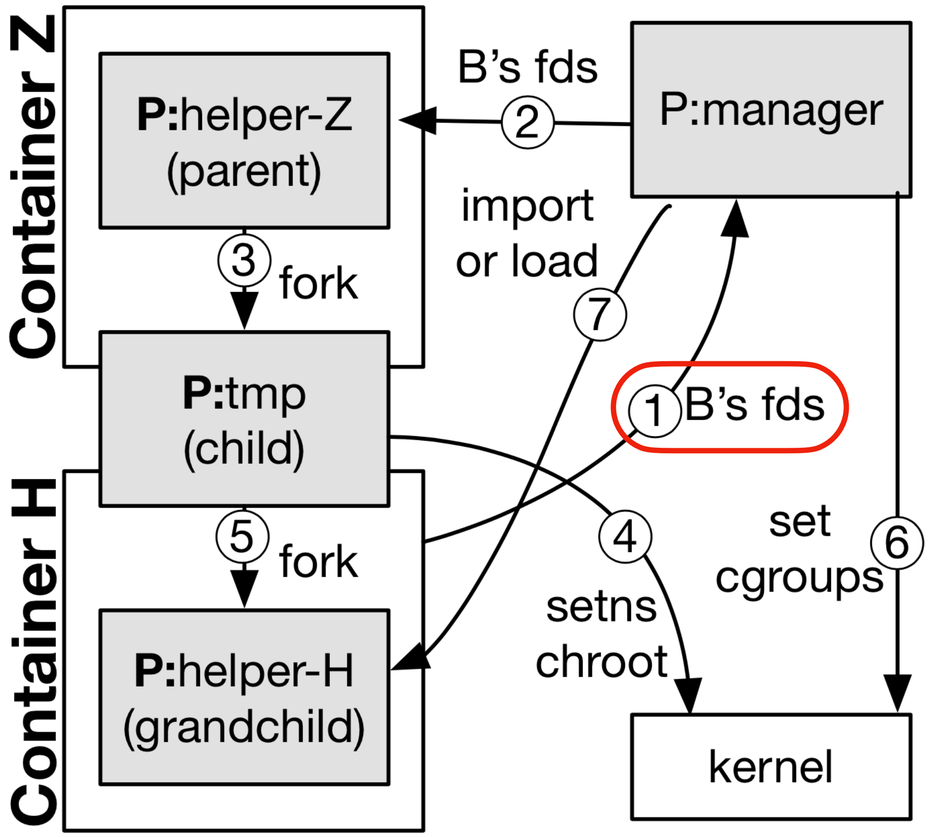
Serverless Impaction

- Download + Install -> Local
- Pre-import a subset of packages because of strong popularity skew

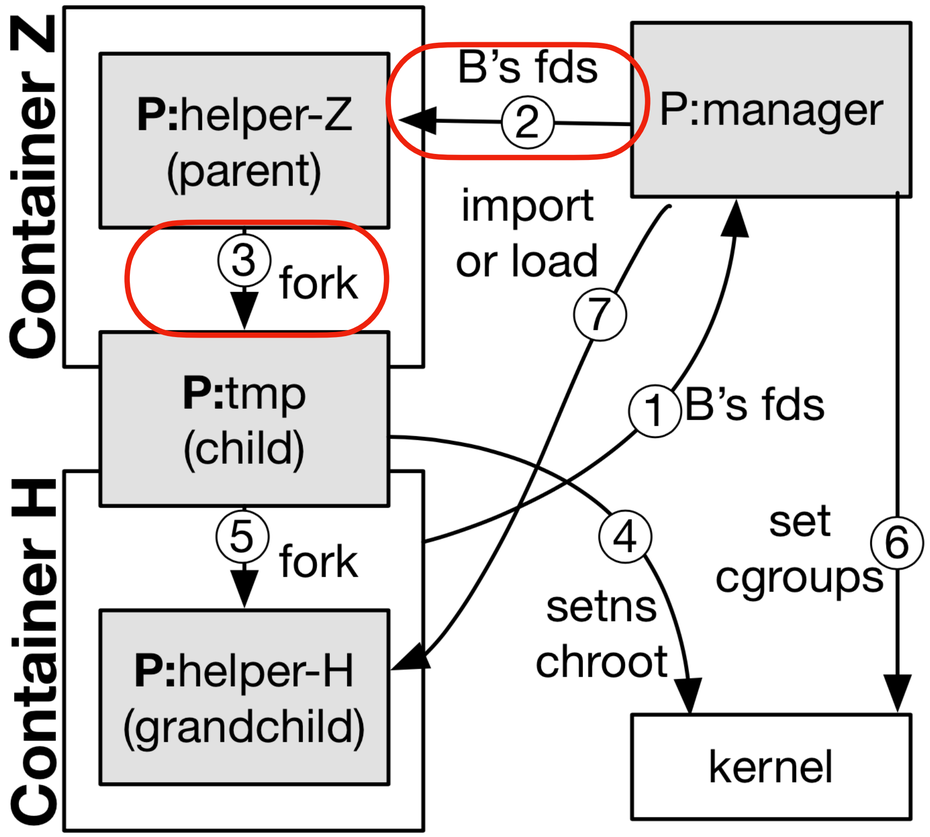
Generalized Zygotes



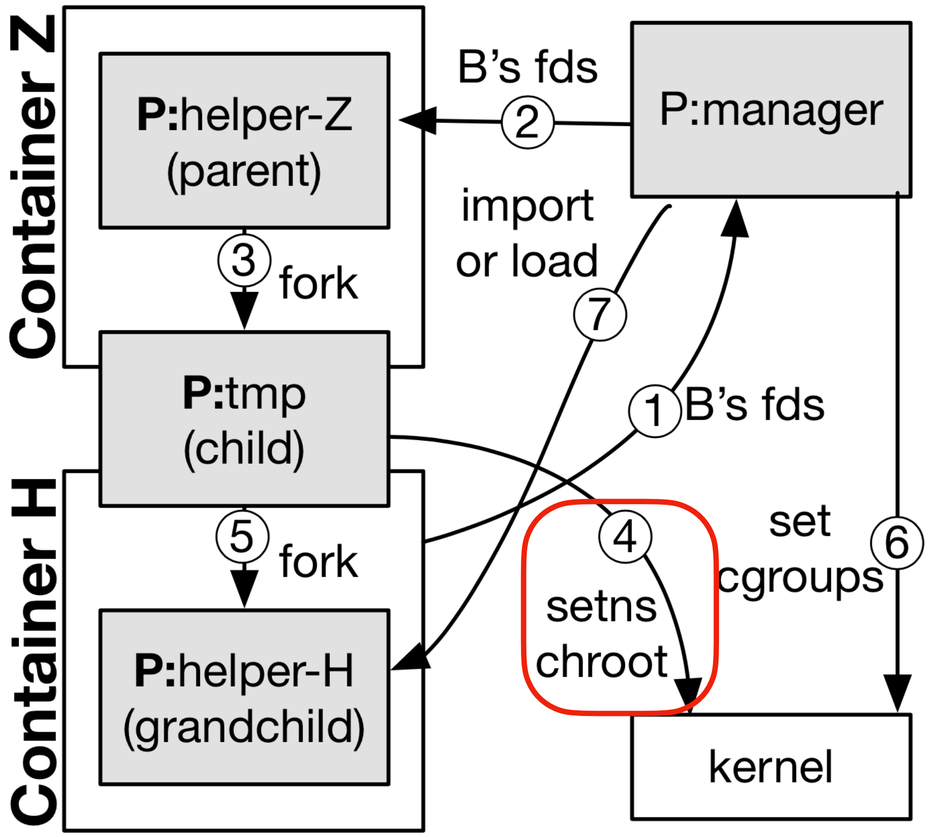
Generalized Zygotes



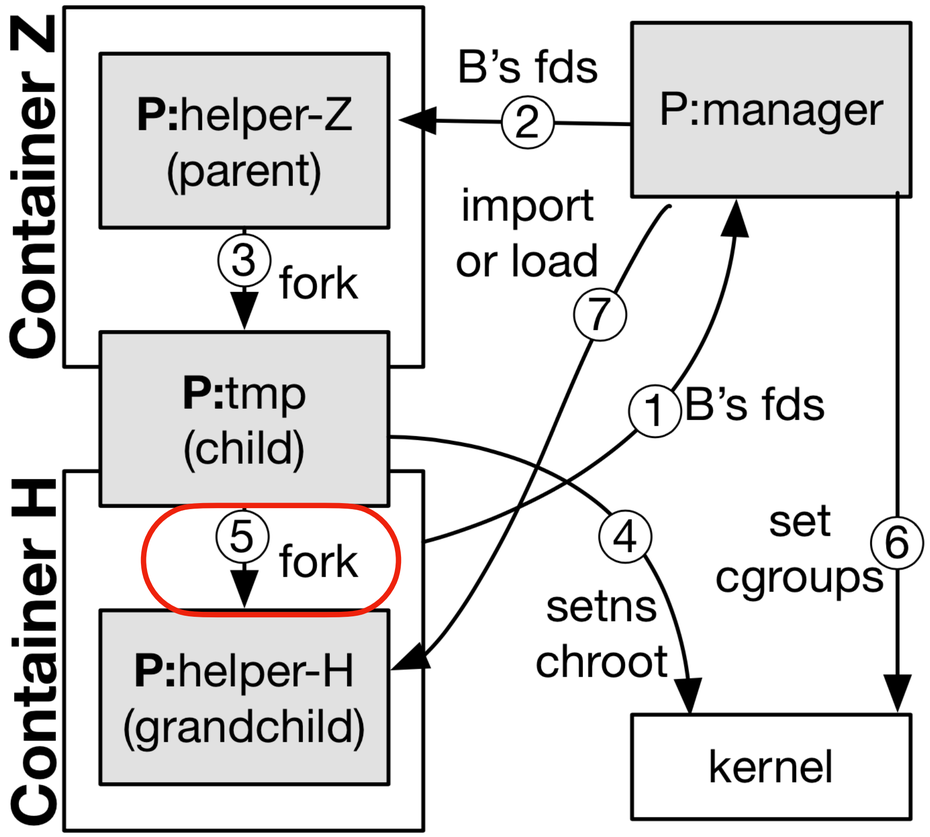
Generalized Zygotes



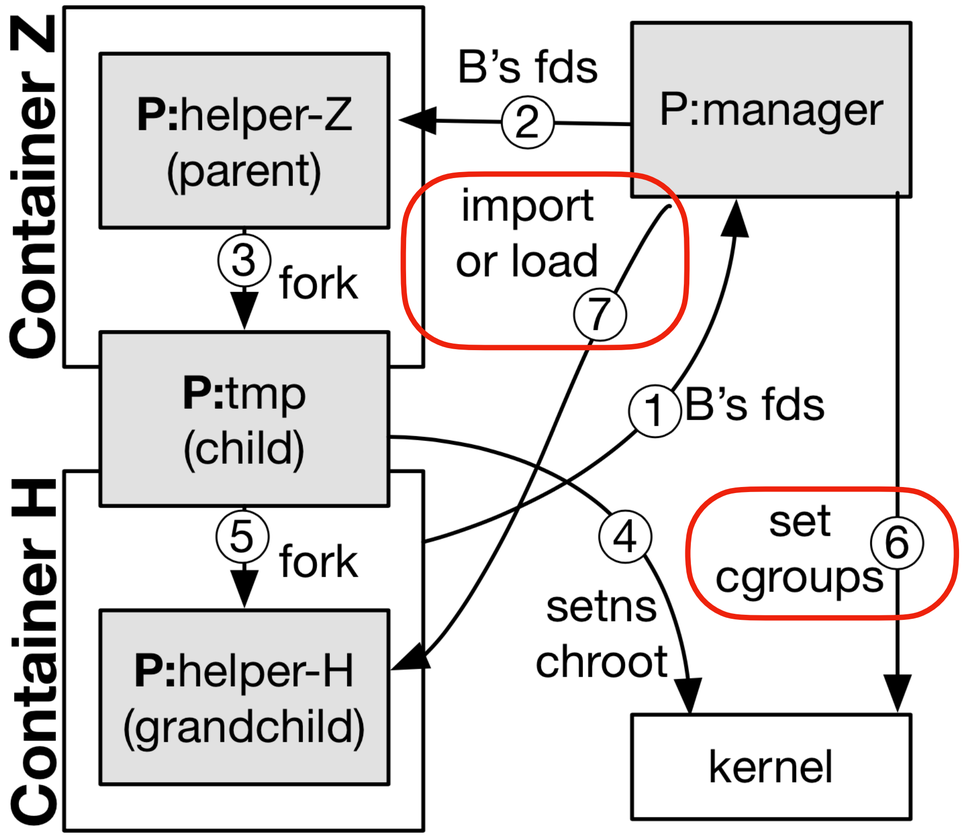
Generalized Zygotes



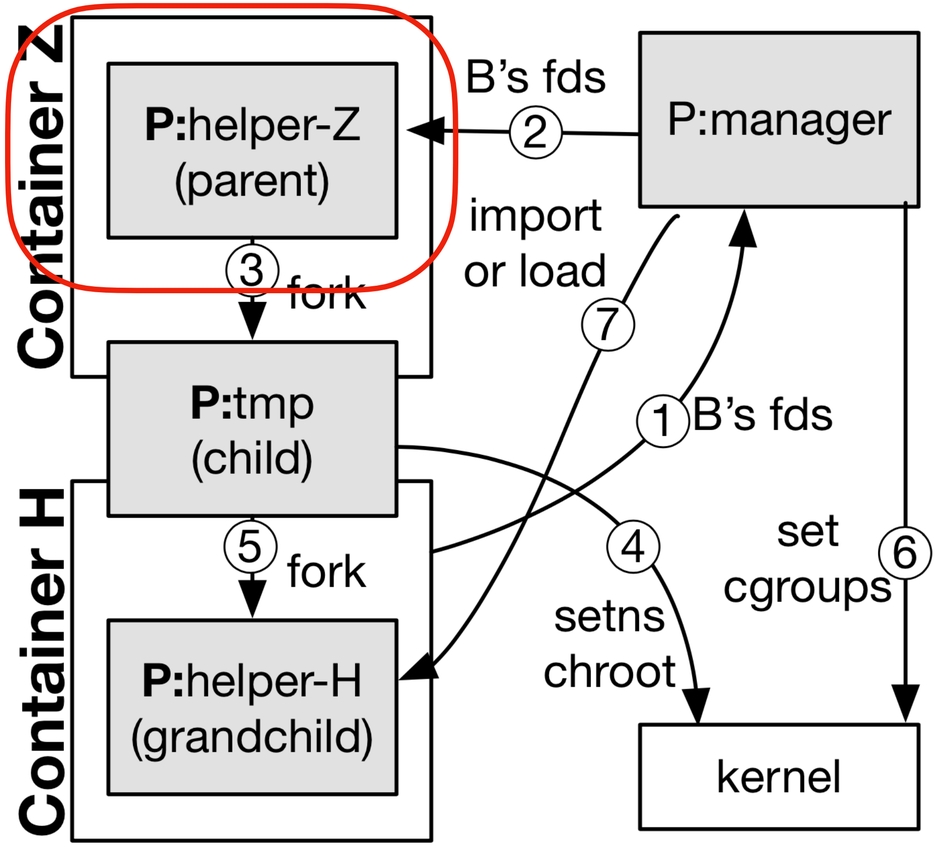
Generalized Zygotes



Generalized Zygotes



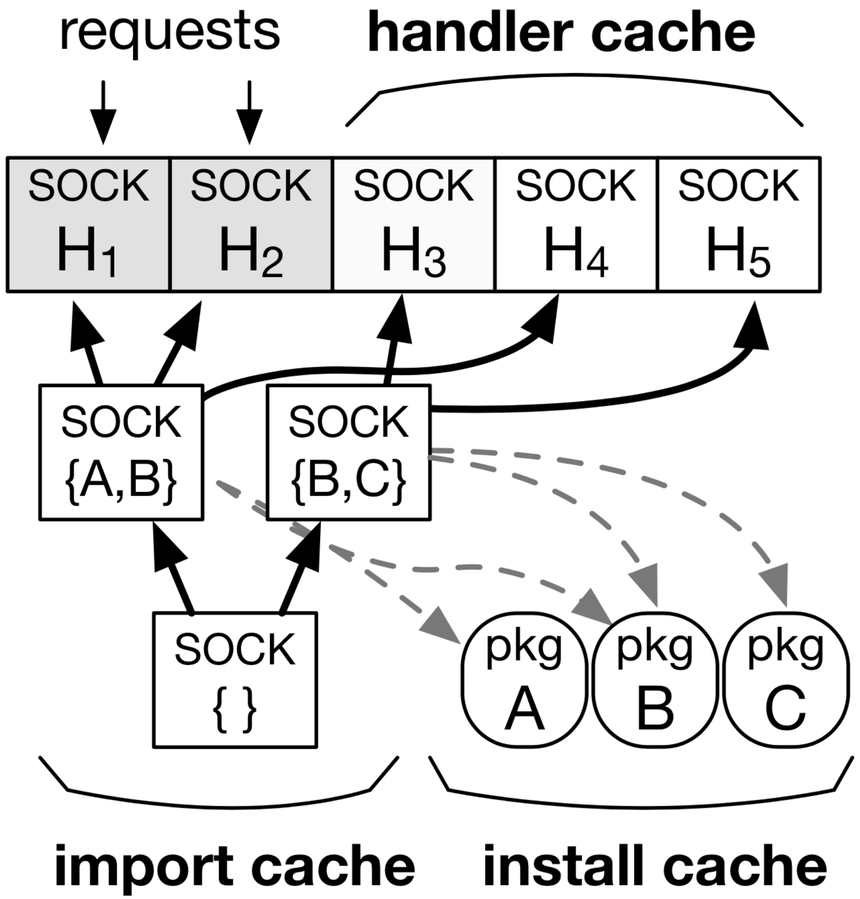
Generalized Zygotes



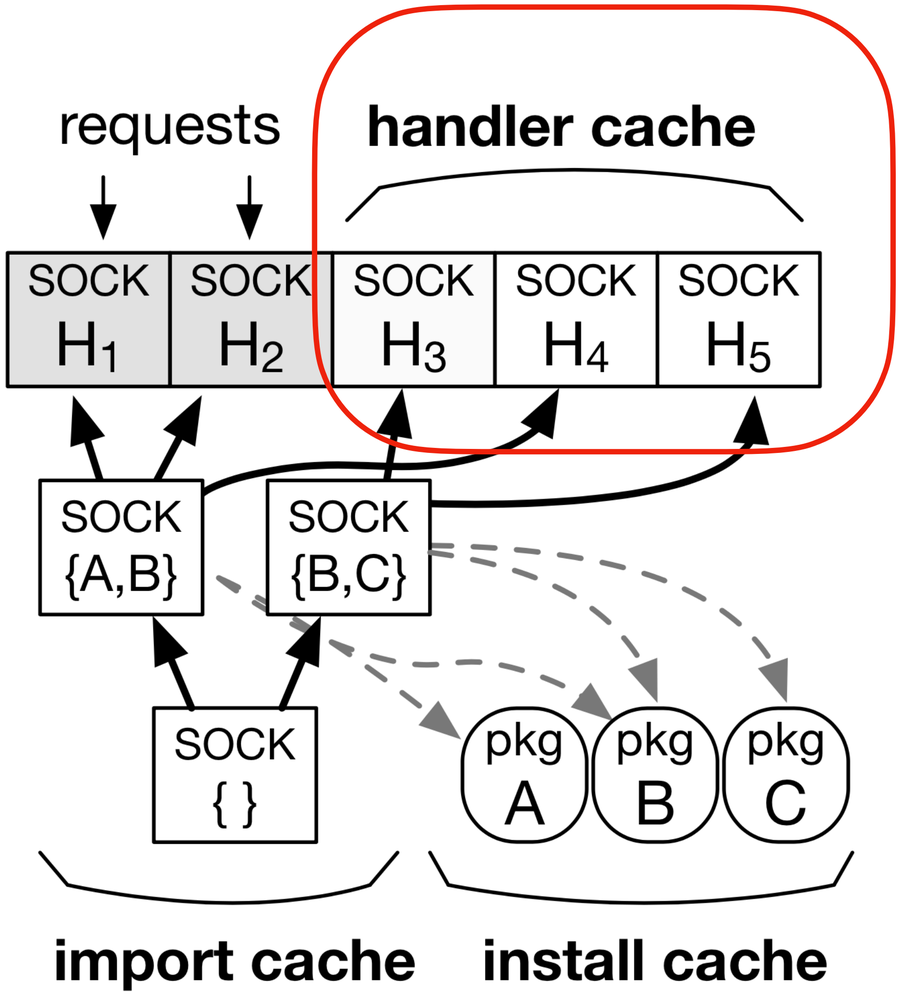
SOCK

- **Lean serverless-optimized containers (SOCK)**
  - Precise usage of Linux isolation mechanisms
  - 18x faster container lifecycle over Docker
- **Provision from secure Zygote processes**
  - Fork from initialized runtime to prevent cold start
  - 3x faster provisioning than SOCK alone
- **Execution caching across 3 tiers**
  - Securely reuse initialization work across customers
  - 3-16x lower platform cost in image-processing case study

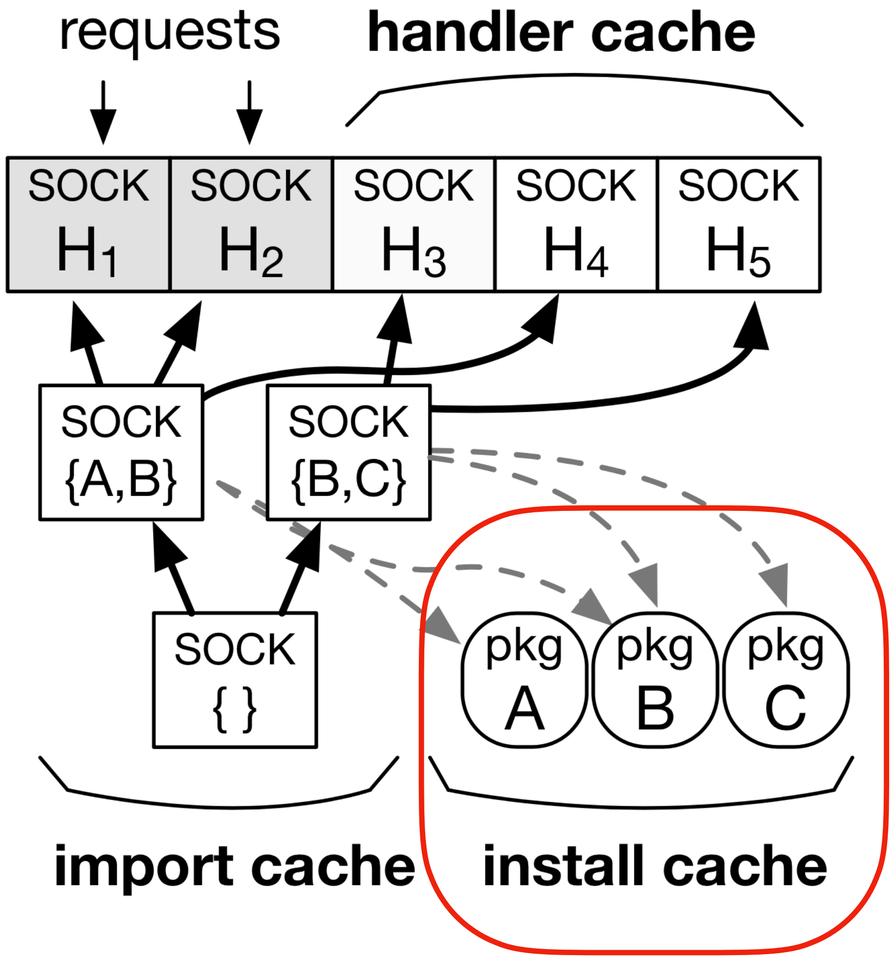
Three-tier Cache



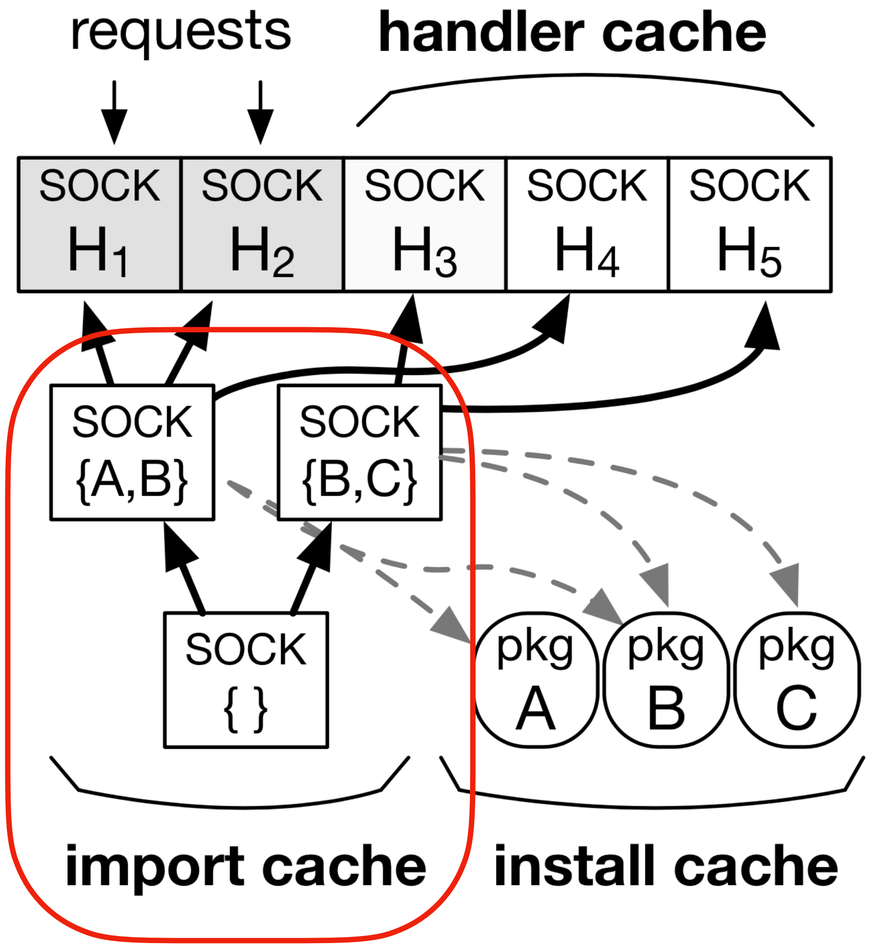
Three-tier Cache



Three-tier Cache



Three-tier Cache



Summary

- **Lean serverless-optimized containers (SOCK)**
- **Provision from secure Zygote processes**
- **Execution caching across 3 tiers**

Thanks