





Poby: SmartNIC-accelerated Image Provisioning for Coldstart in Clouds

***Zihao Chang*[1,2]**, Jiaqi Zhu[1,2], Haifeng Sun[3] , Yunlong Xie[1,2], Kan Shi[1,2] , Ninghui Sun[1,2], Yungang Bao[1,2], Sa Wang[1,2]

[1] SKLP, Institute of Computing Technology, CAS
[2] University of Chinese Academy of Sciences
[3] Peking University

ATC 2025 · Boston, MA, USA
July 7, 2025

Coldstart in Clouds

- Coldstart is one of the classic challenges that plague cloud platform
  - Orders of magnitude higher latency penalty

- Serverless computing further exacerbates this problem
  - Higher elasticity and lower latency requirements







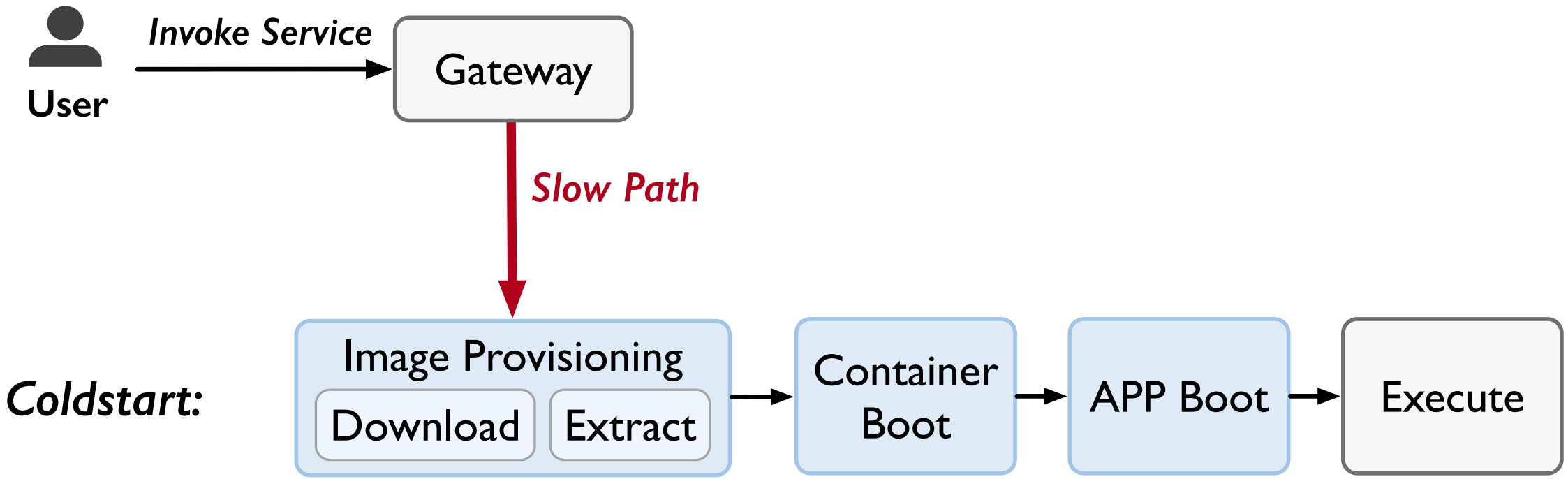




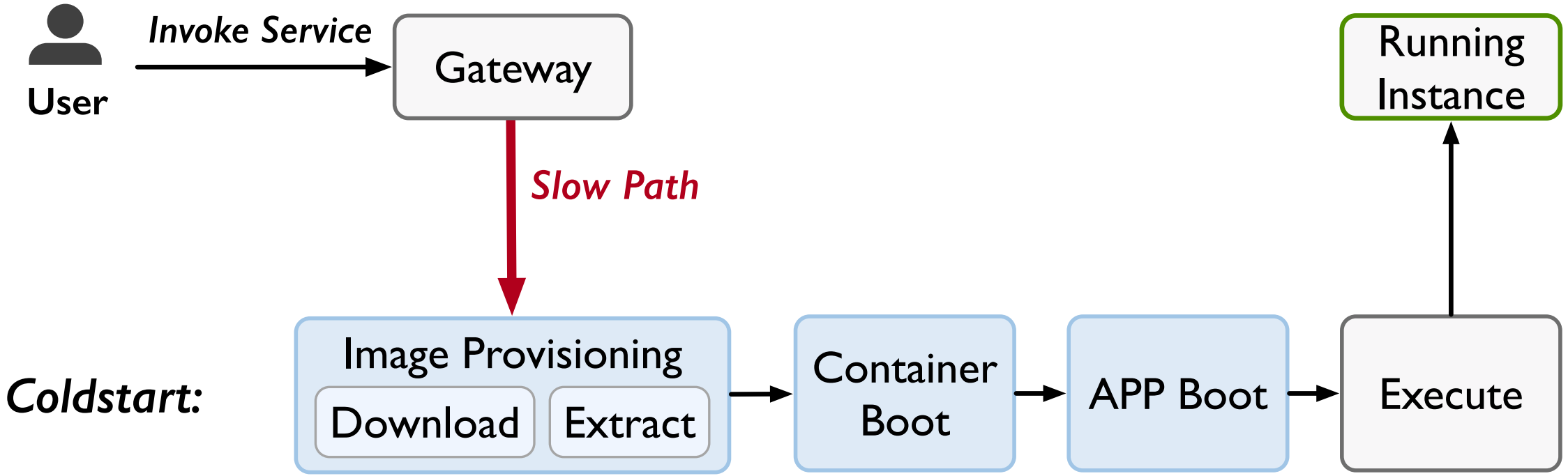
Workflow of Coldstart



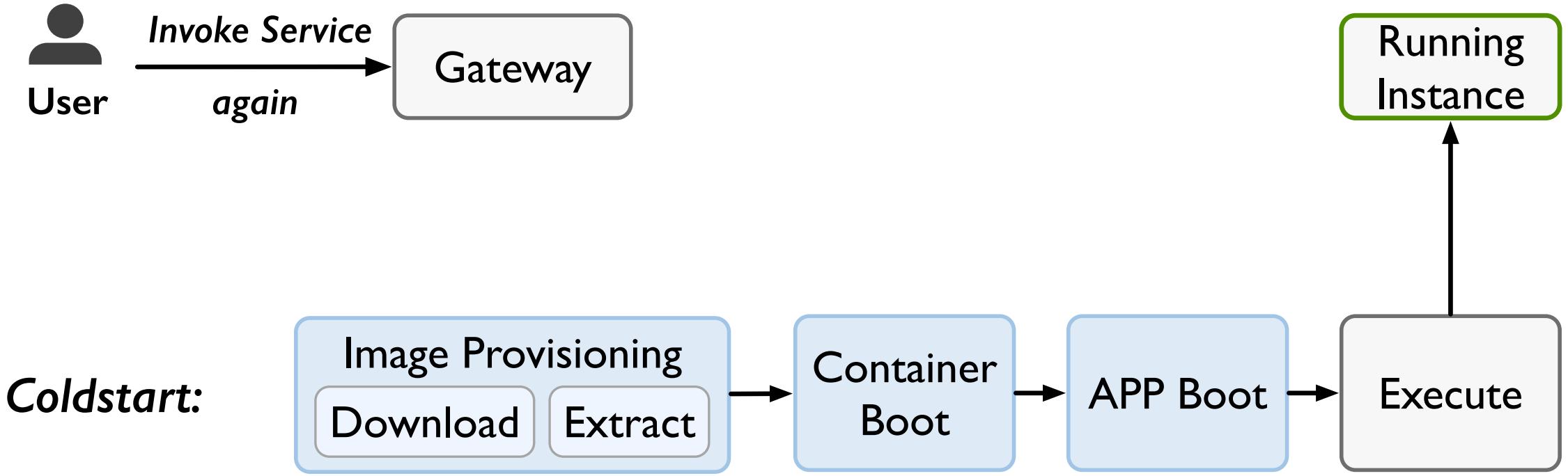
Workflow of Coldstart



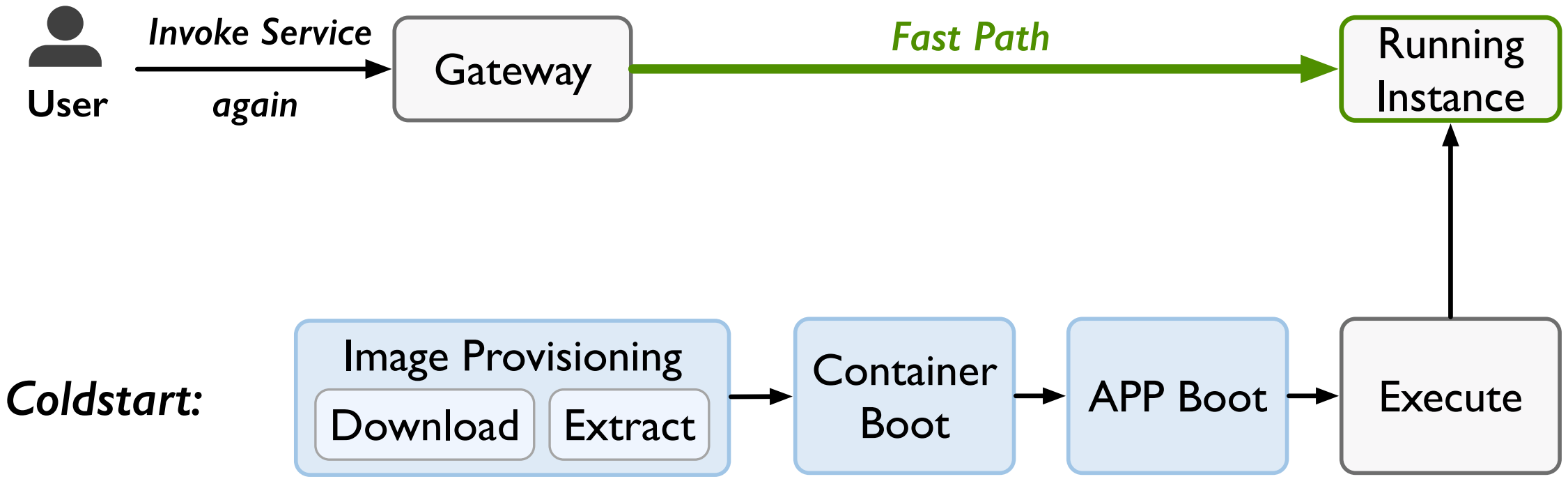
Workflow of Coldstart



Workflow of Warmstart



Workflow of Warmstart



The Entire Workflow

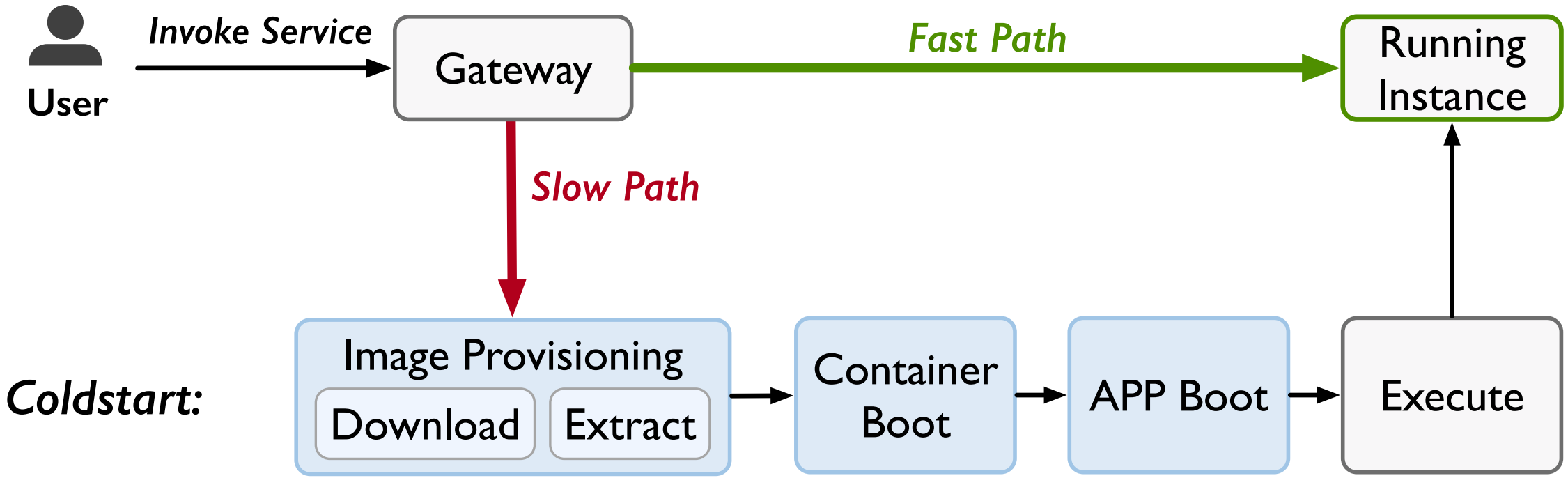
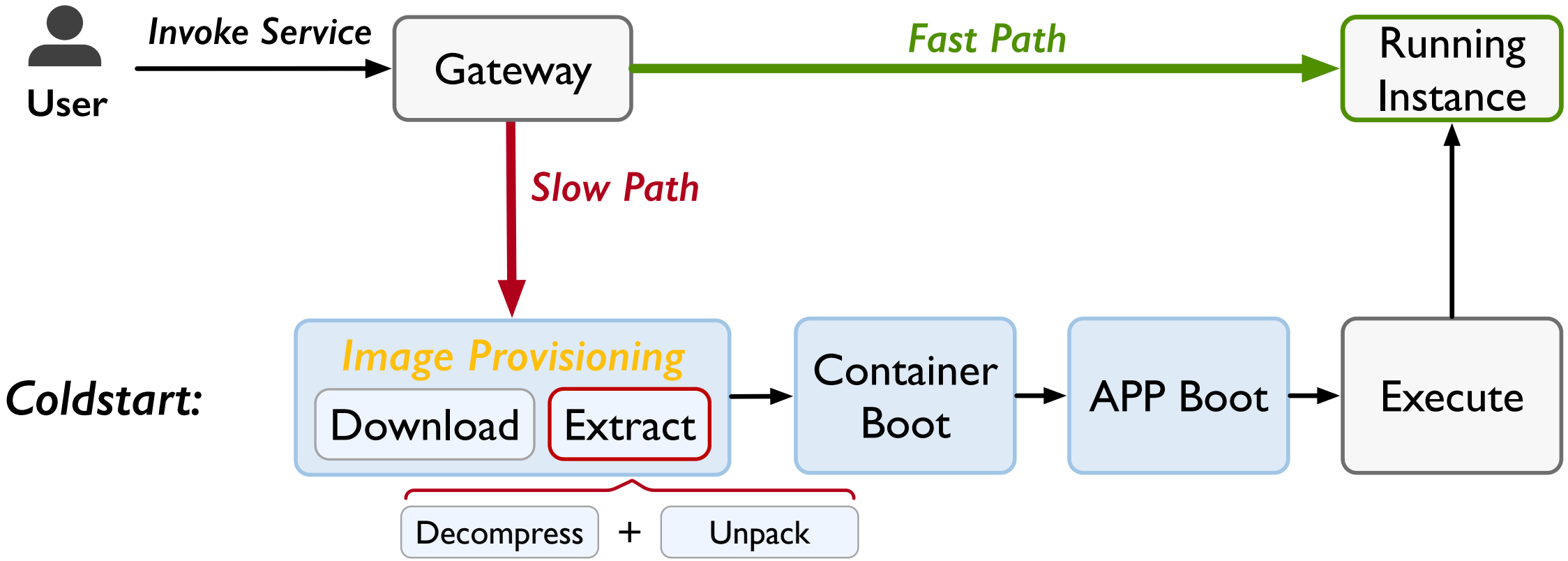


Image Provisioning



Observation #1: Time-consuming Step

① *Image provisioning* is the *most time-consuming* step of coldstart

- ***57%+*** of coldstarts take ***>45s***
- ***50%+*** function requests spend ***>72%*** of time on image provisioning

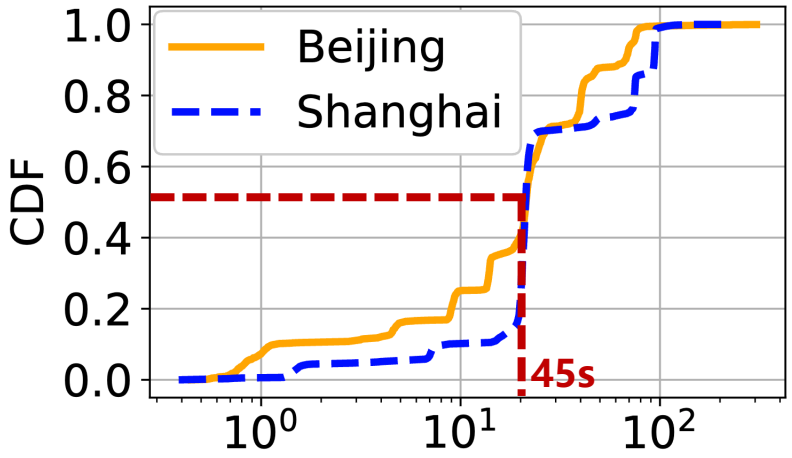
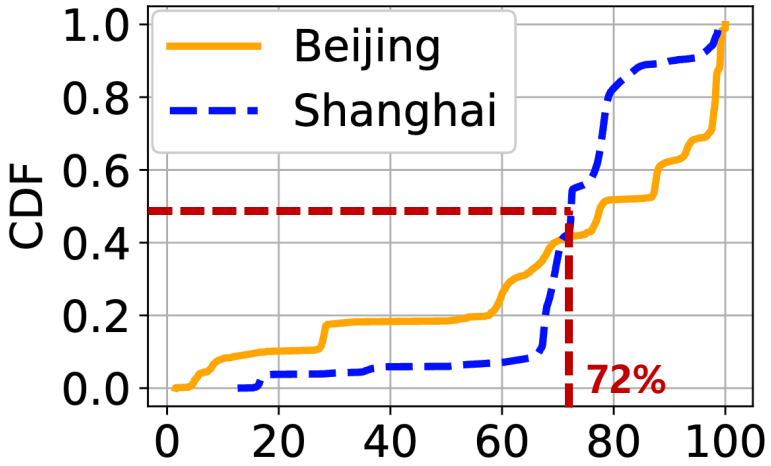


Image provisioning latency (s)



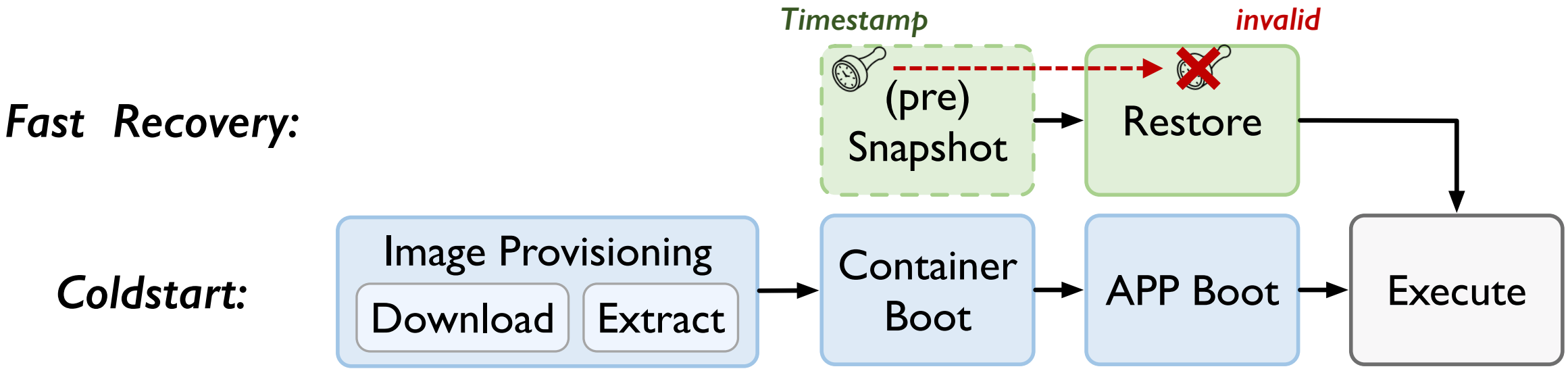
% of image provisioning in coldstart

712,295 coldstarts of functions within 15 days from Alibaba Cloud [2]

[2] Wang, Ao, et al. "{FaaSNet}: Scalable and fast provisioning of custom serverless container runtimes at alibaba cloud function compute." *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 2021.

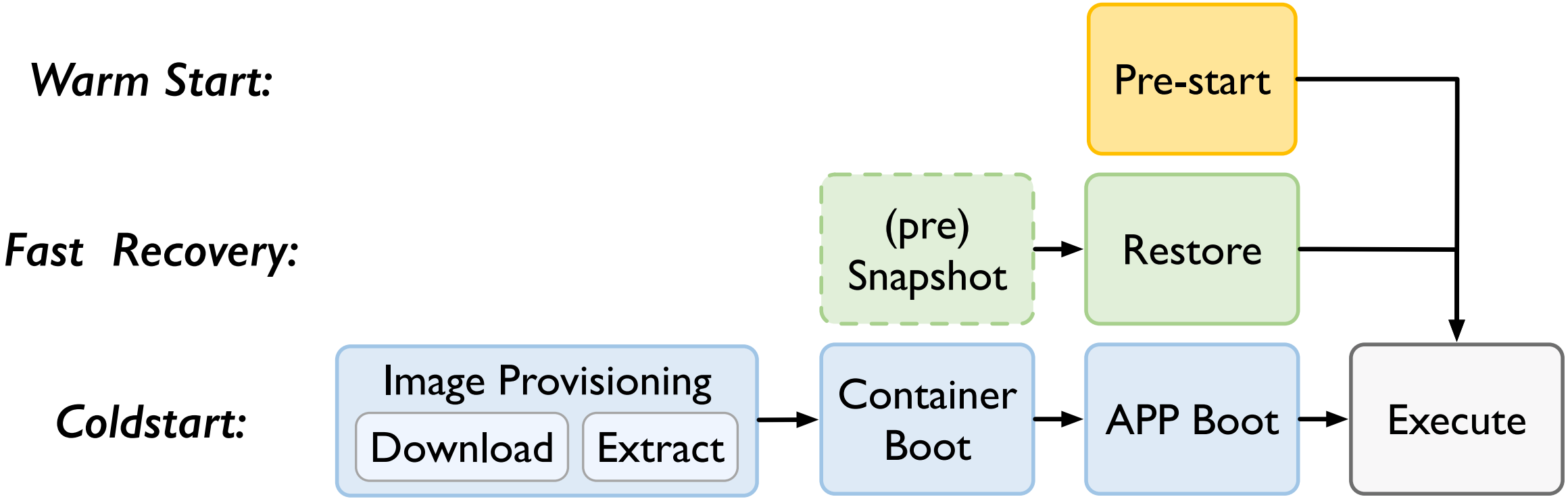
Current Solution #1: Fast Recovery

① Significant overhead for *proactively maintaining* snapshot
② Only suitable for *stateless* applications



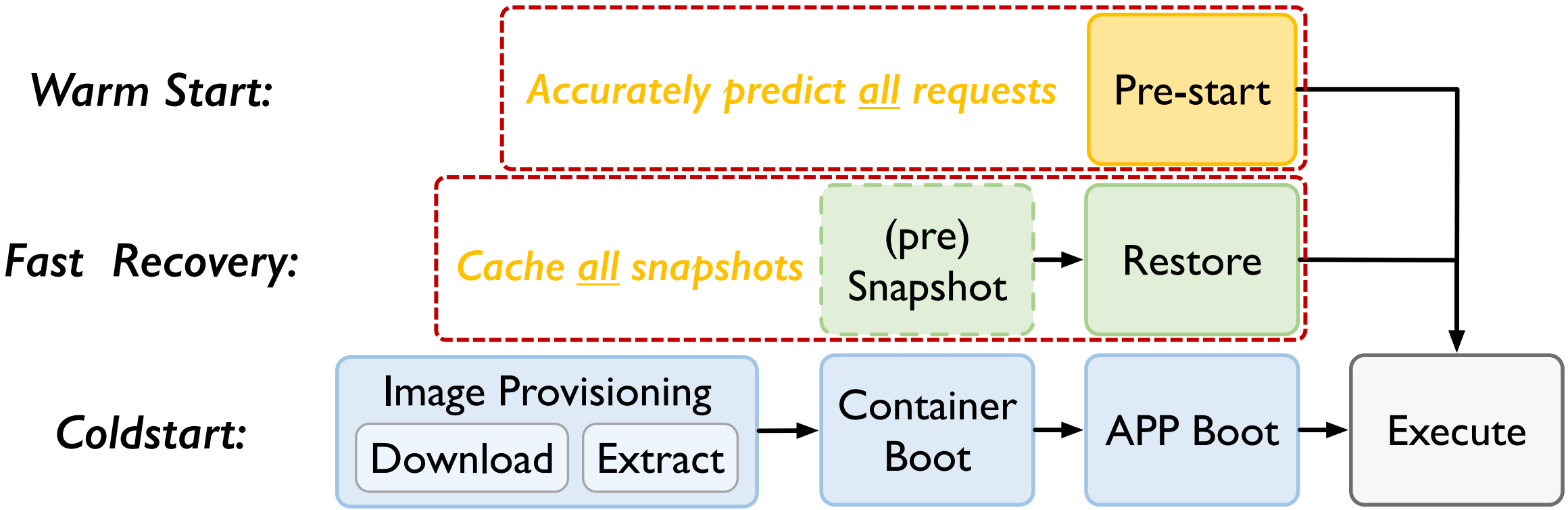
Current Solution #2: Warm Start

① Keep-alive overhead even *without requests*
② Not work for *new deployed* containers



Limitations of Avoidance Methods

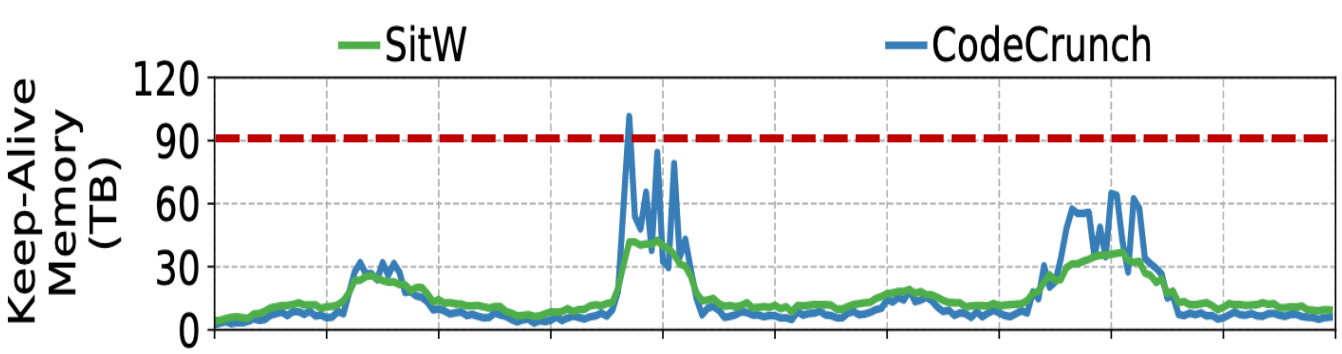
These methods place *extremely high demands* on *pre-profile*

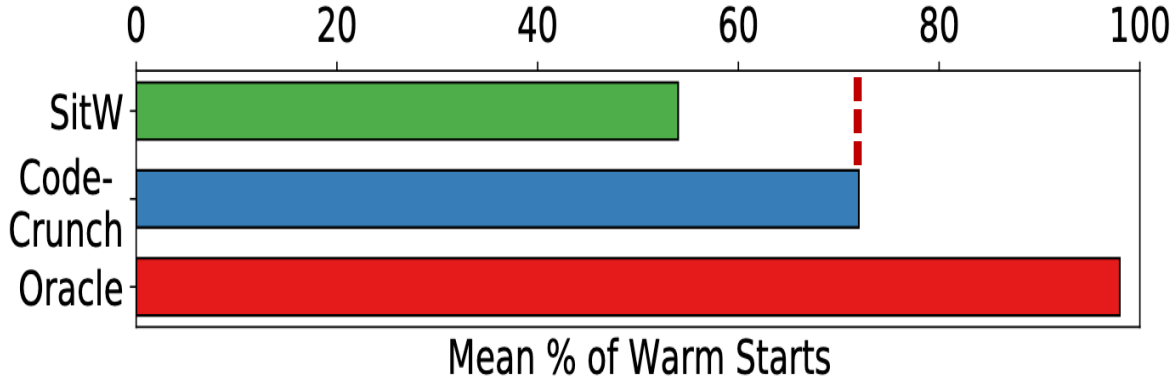


Observation #2: Inevitable

② Coldstart from image provisioning is *inevitable*

- It is not cost-effective to snapshot all containers
- It is impossible to accurately predict the arrival of requests



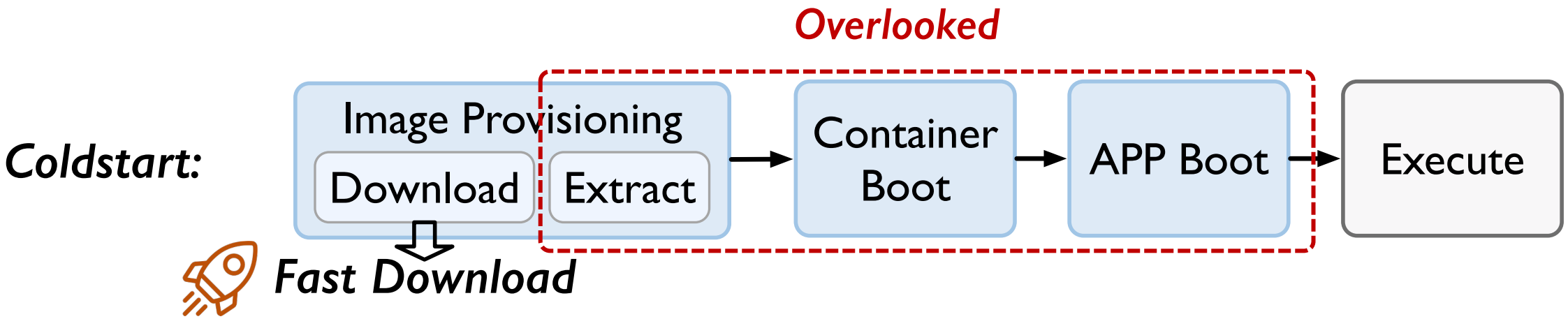


CodeCrunch [1], a SOTA warm start work, achieves <80% warm start rate with >90 TB peak memory usage

[1] Basu Roy, Rohan, et al. "Codecrunch: improving serverless performance via function compression and cost-aware warmup location optimization." *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 2024.

Current Solution #3: Fast Download

① Significant image *proactively cache* overhead
② New image formats cause *compatibility* issues
③ Partial optimization on a *single* operation

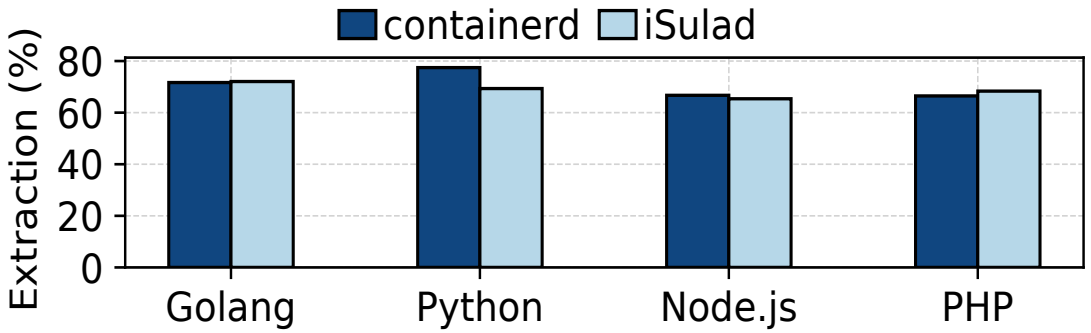


Observation #3: Overlooked Key Step

③ Recent efforts ignore the *image extraction* process

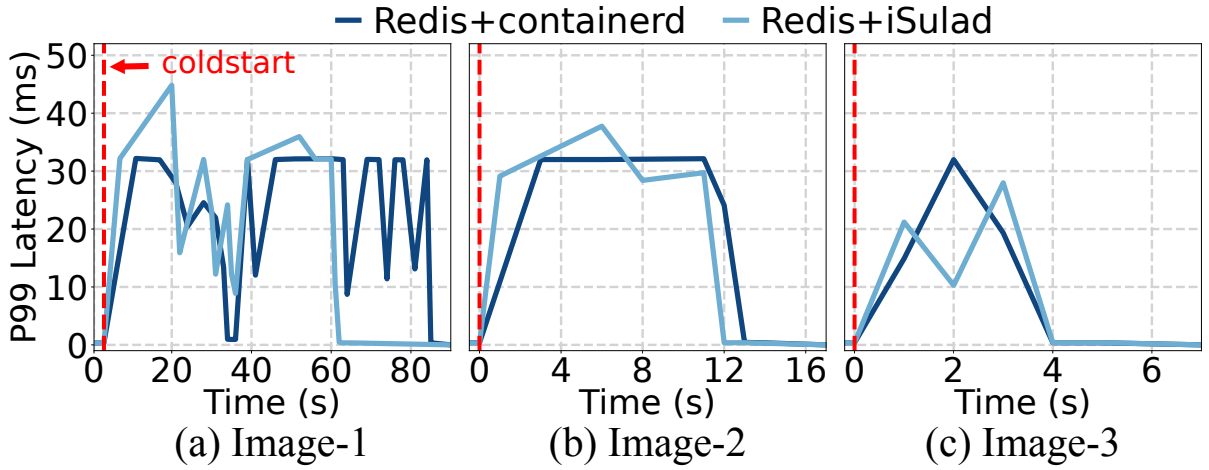
- Extraction takes most of the image provisioning time ***(bottleneck)***
- Introduce severe performance interference to colocated apps ***(interference)***

Extraction takes *>68.8%* time



% of extraction time in image provisioning

The latency of Redis increases *>110×*

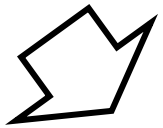


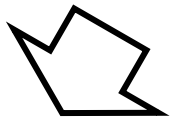
Tail latency of Redis when colocated with coldstart

Motivation

Serverless computing further *exacerbates* the coldstart issue

Existing methods *fail* to effectively address the problem





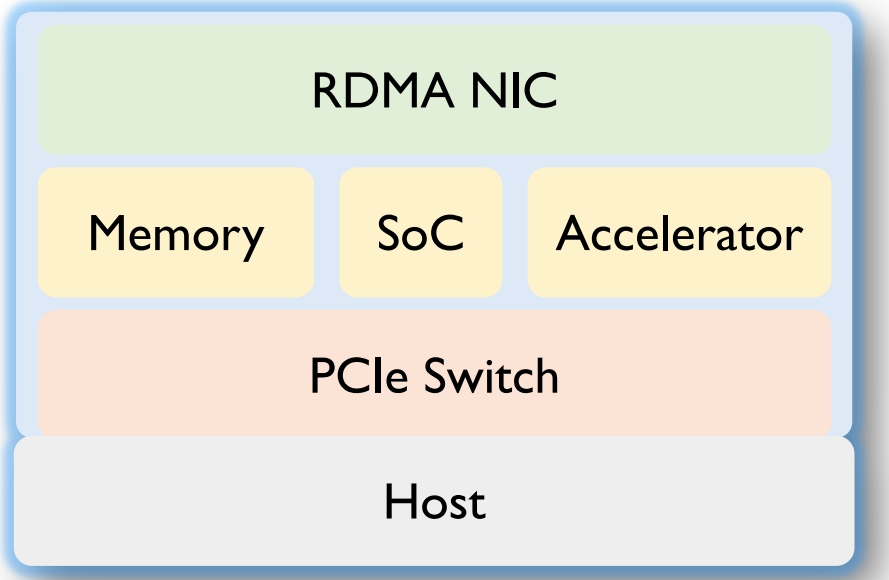
Directly accelerate the _entire_ coldstart process and _completely_ solve the problem

Design

Key Idea

SmartNIC enables Network-wide collaborative coldstart offloading





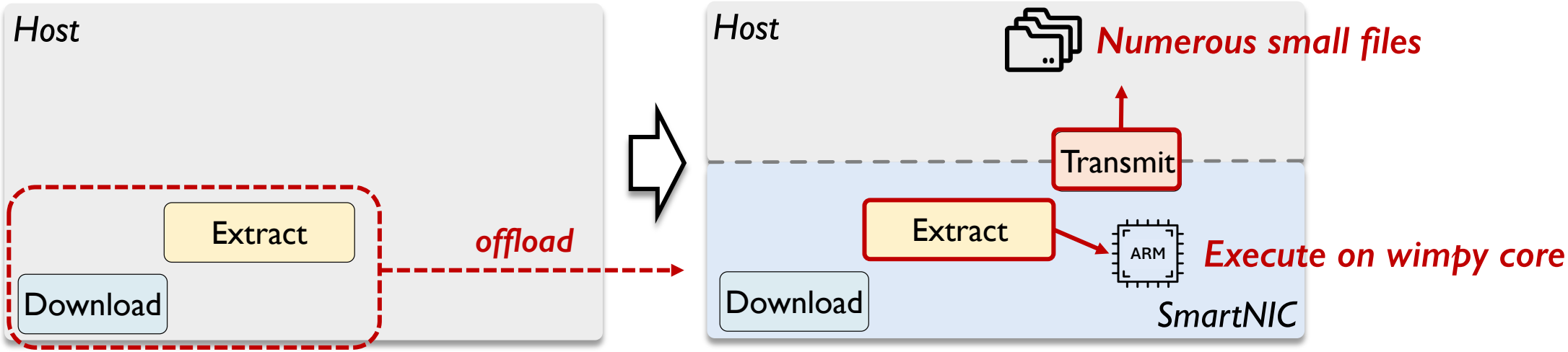
- Network: low latency and high-bandwidth RDMA (up to 400 Gbps)
- On-NIC computing power: hardware accelerators and general-purpose SoC

Putting this idea into practice still faces many *challenges*

Challenges I: Offloading Dilemma

✖ **Simply offloading the entire coldstart to SmartNIC is *inefficient***

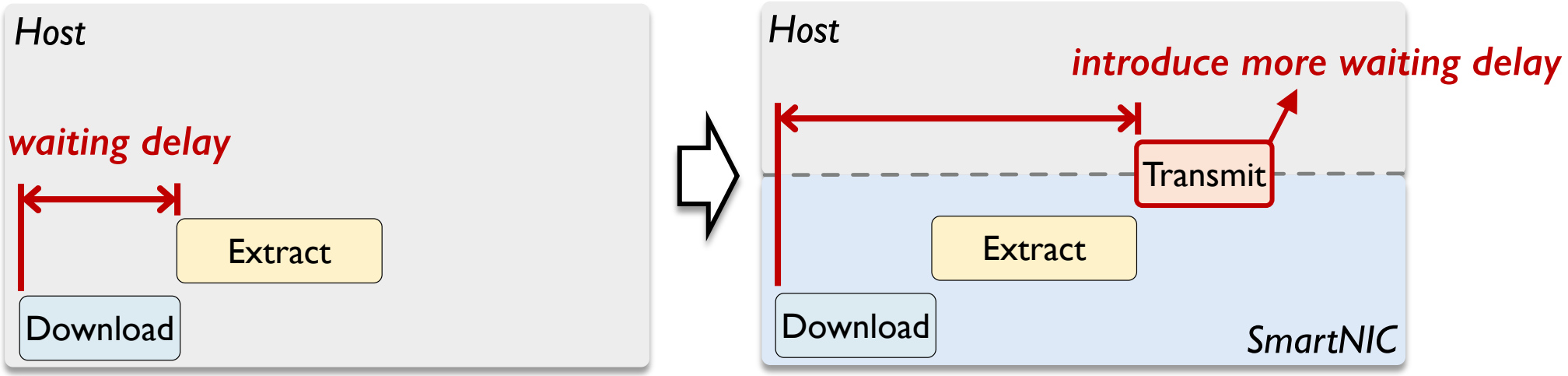
- Computing power of on-NIC SoC is wimpy
- Introduce transmission of numerous extracted files



Challenges II: Serial Execution

✖ **The store-and-forward workflow results in *significant waiting delay***

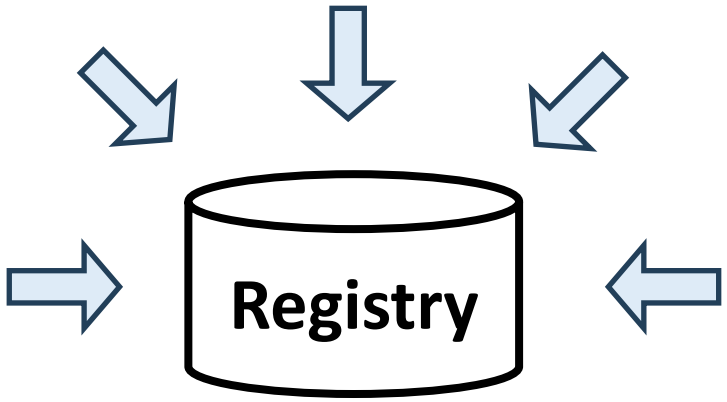
- Each operation suffer a delay of waiting for the previous operation to complete
- Additional operations increase waiting delays



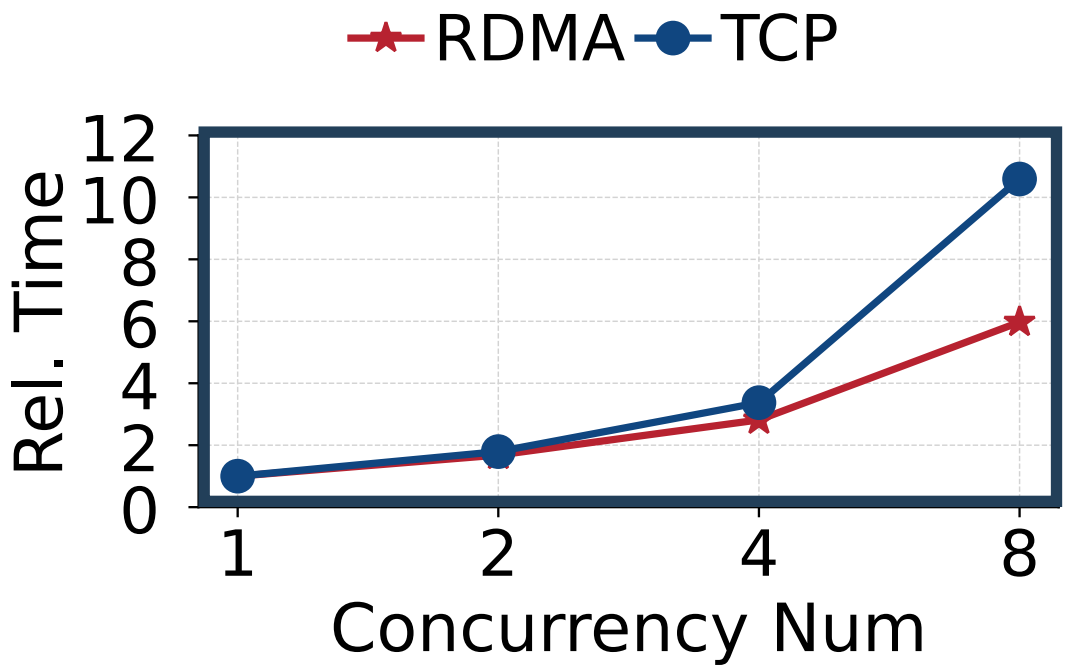
Challenges III: Centralized Image Registry

✖ **Centralized image registry becomes *bottleneck* when coldstart burst**

- Concurrent image downloads cause severe bandwidth competition



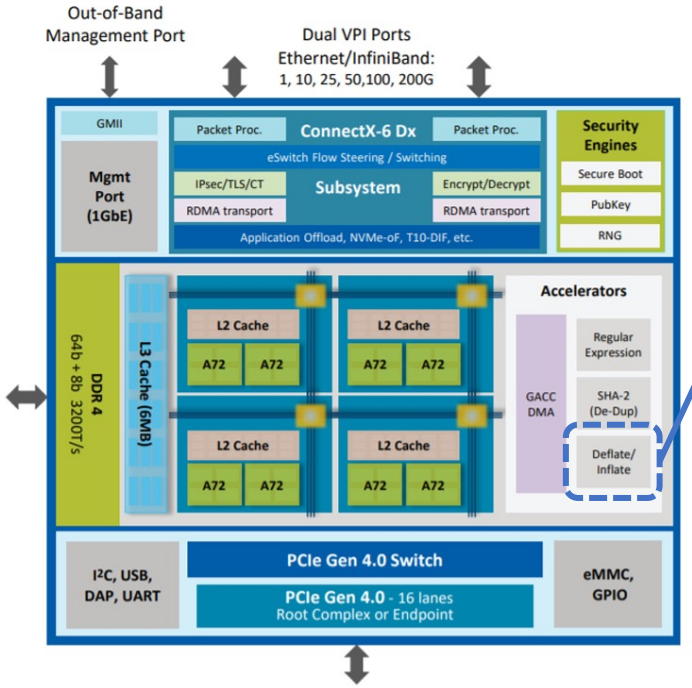
Competition of coldstarts



Design I: Disaggregated Architecture

✔ **Orchestrate resources of the *entire cluster* to accelerate image provisioning**

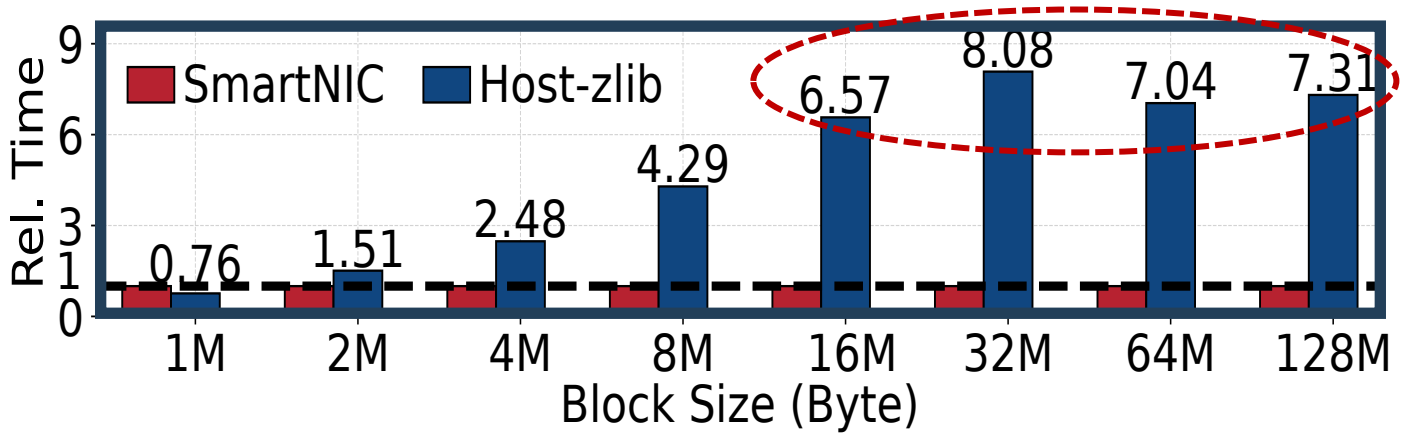
- Utilize hardware to accelerate decompression



Architecture of Bluefield-2

Hardware decompression accelerator

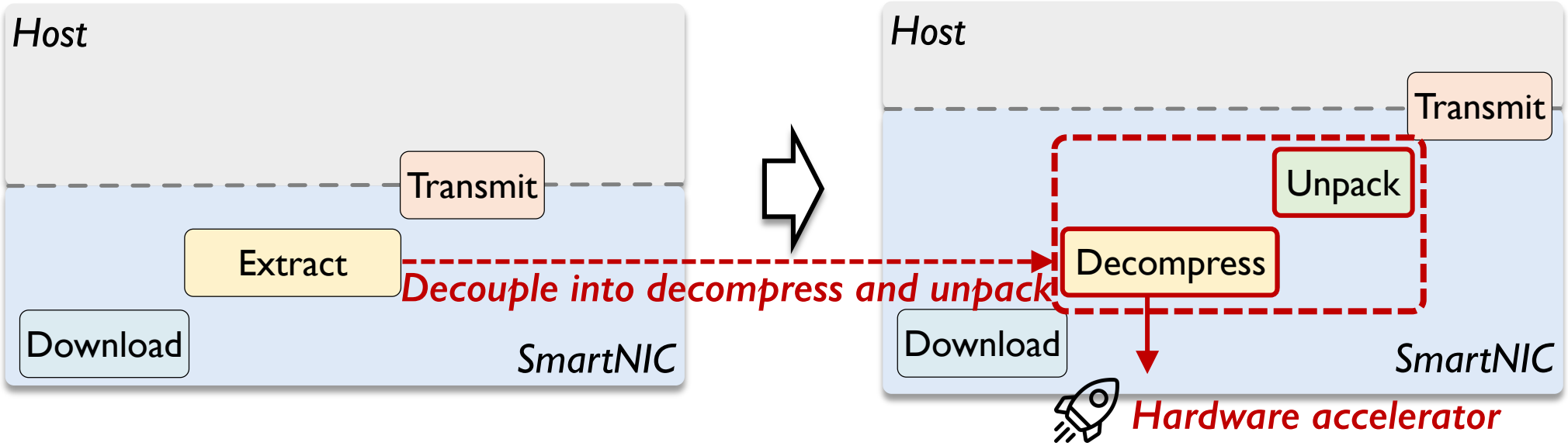
Reduce more than 80% decompression time



Design I: Disaggregated Architecture

✔ **Orchestrate resources of the *entire cluster* to accelerate image provisioning**

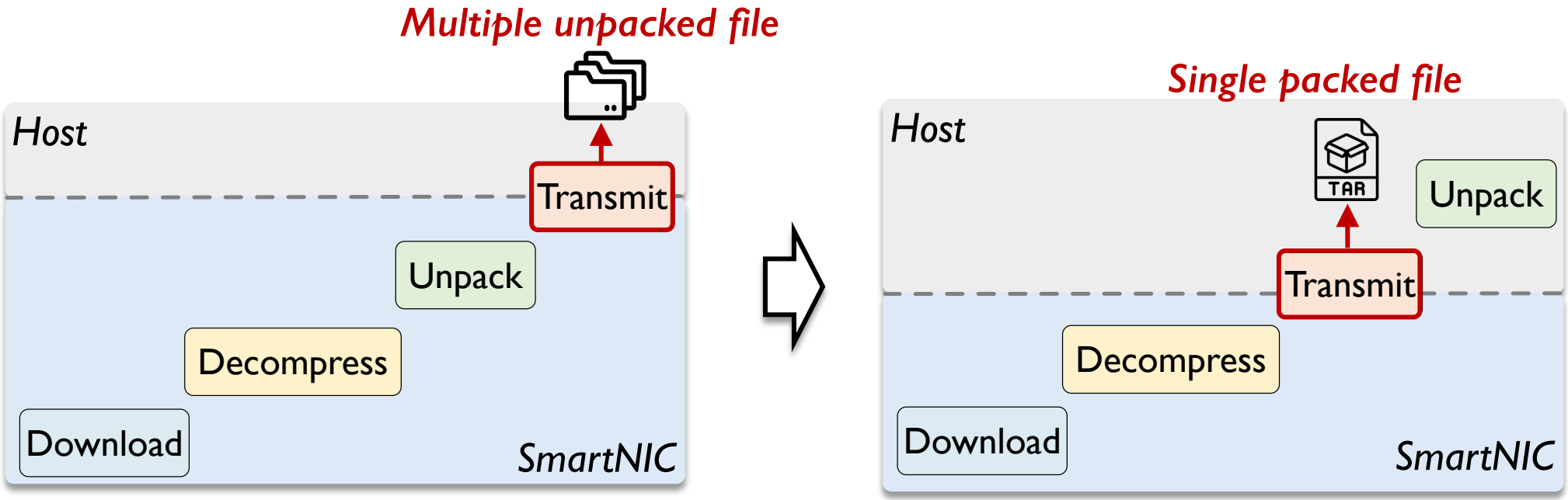
- Utilize hardware to accelerate decompression



Design I: Disaggregated Architecture

✔ **Orchestrate resources of the *entire cluster* to accelerate image provisioning**

- Utilize hardware to accelerate decompression
- New execution order bypassing transmission bottleneck

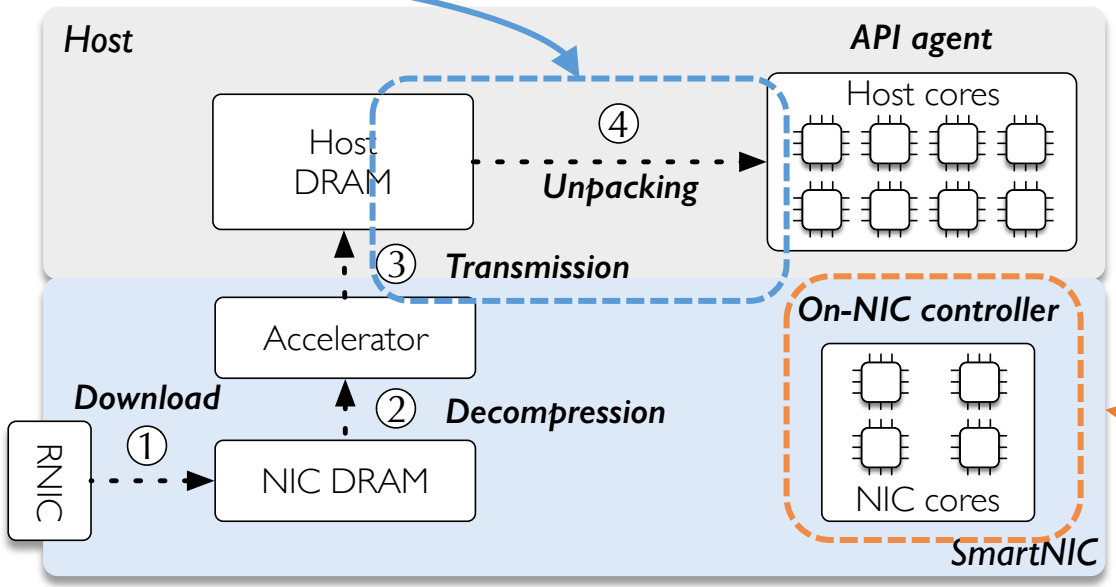


Design I: Disaggregated Architecture

✔ **Orchestrate resources of the *entire cluster* to accelerate image provisioning**

- Utilize hardware to accelerate decompression
- New execution order bypassing bottleneck

Transfer a single packed file is much more efficient

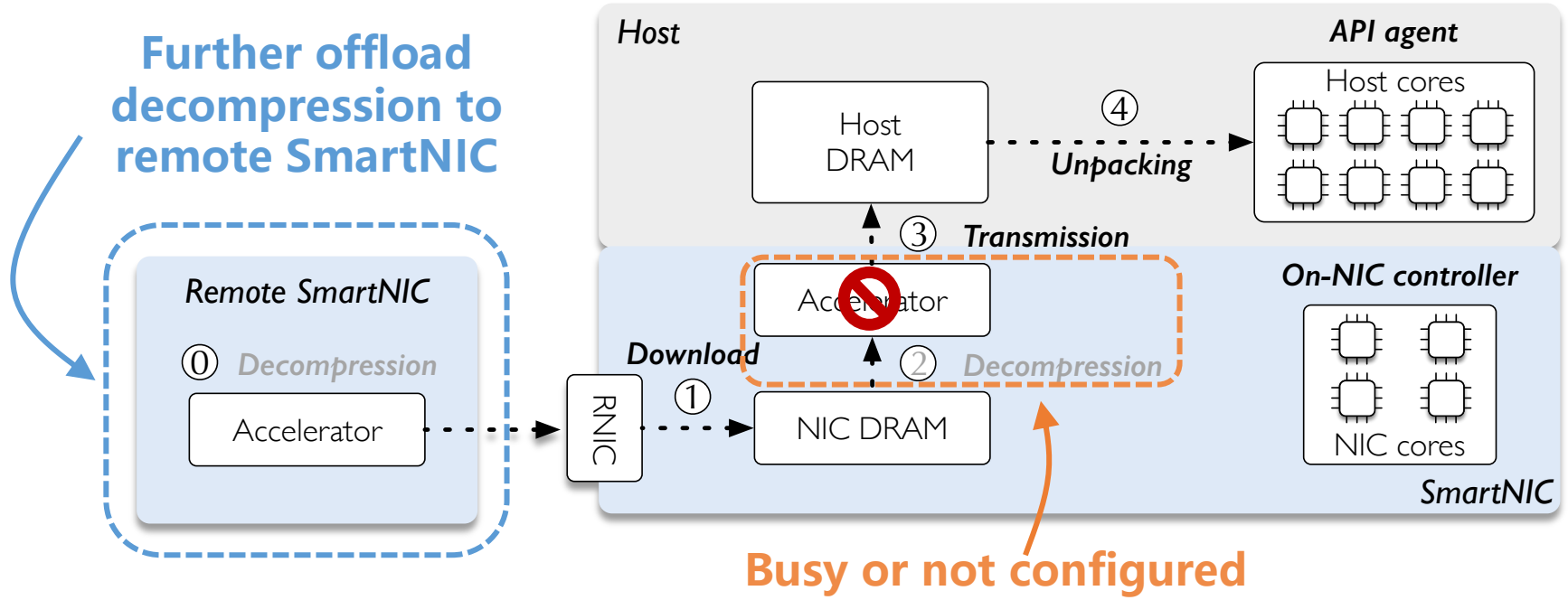


Avoid overloading the on-NIC SoC with unpack operation

Design I: Disaggregated Architecture

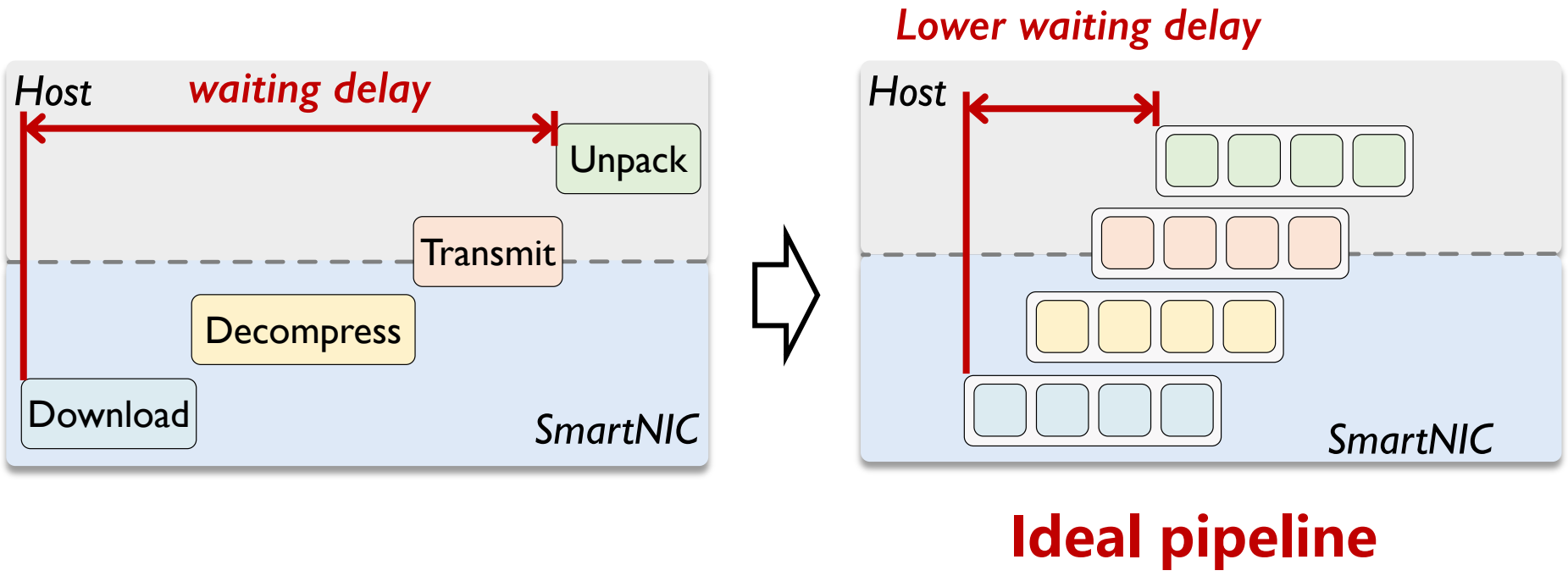
✔ **Orchestrate resources of the *entire cluster* to accelerate image provisioning**

- Utilize hardware to accelerate decompression
- New execution order bypassing bottleneck
- Network-wide collaborative enhancement



Design II: Pipeline-based Data-driven Workflow

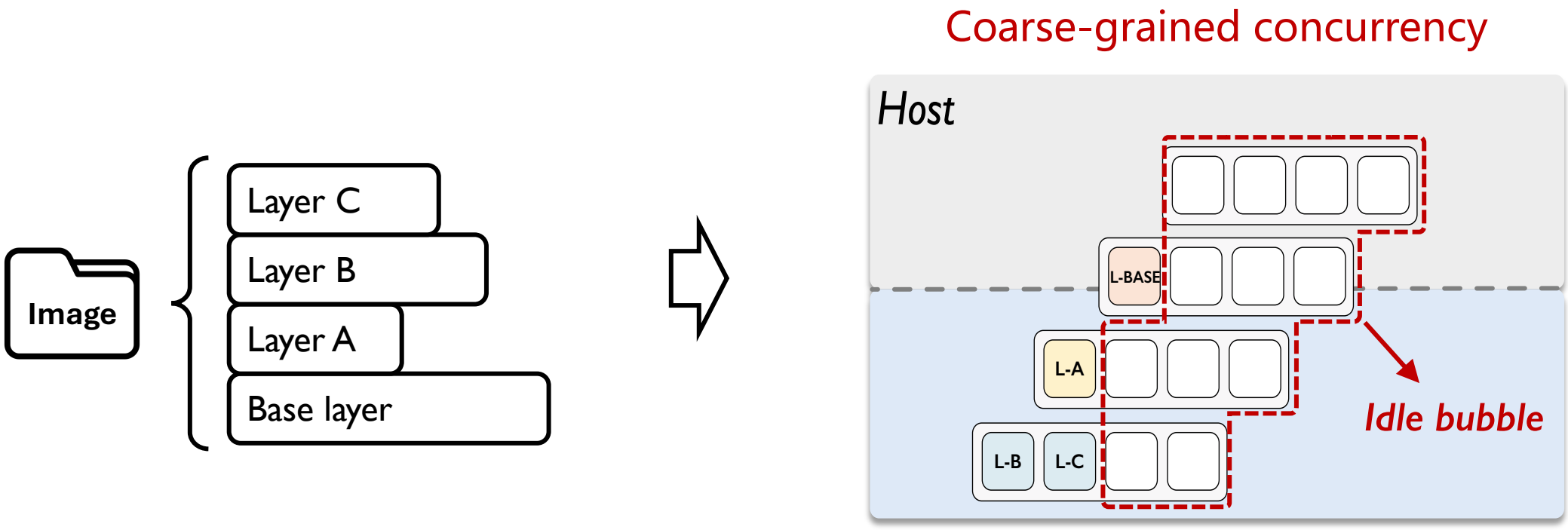
✔ **Pipeline the workflow to enhance *efficiency***



Design II: Pipeline-based Data-driven Workflow

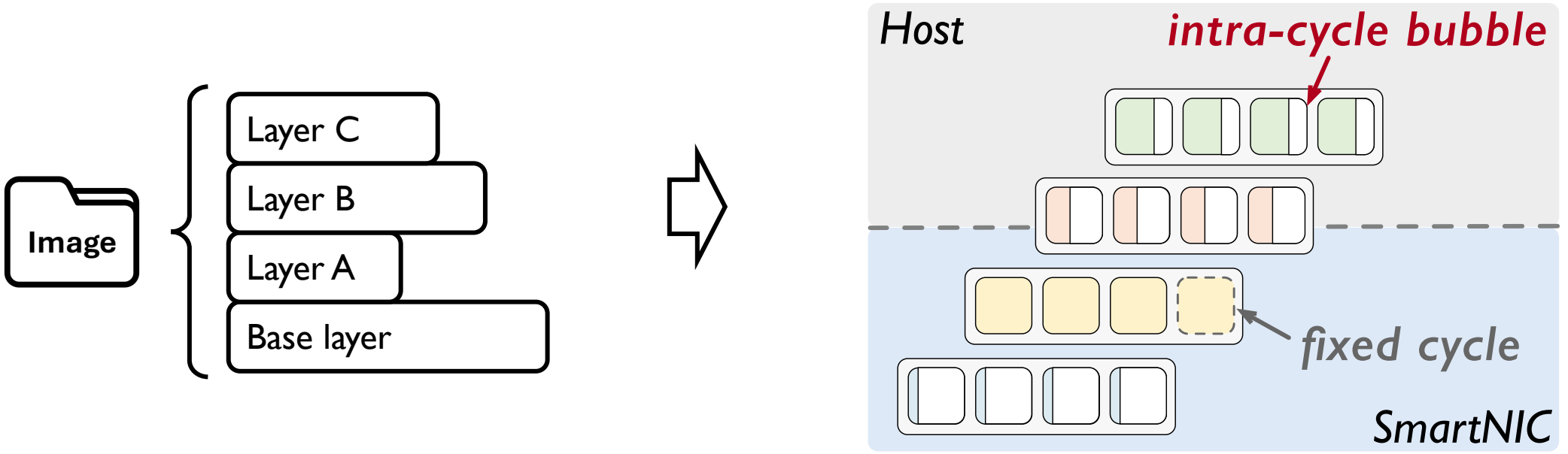
✔ **Pipeline the workflow to enhance *efficiency***

✘ Layer-based pipelines ⇒ Limited parallelism & idle bubble



Design II: Pipeline-based Data-driven Workflow

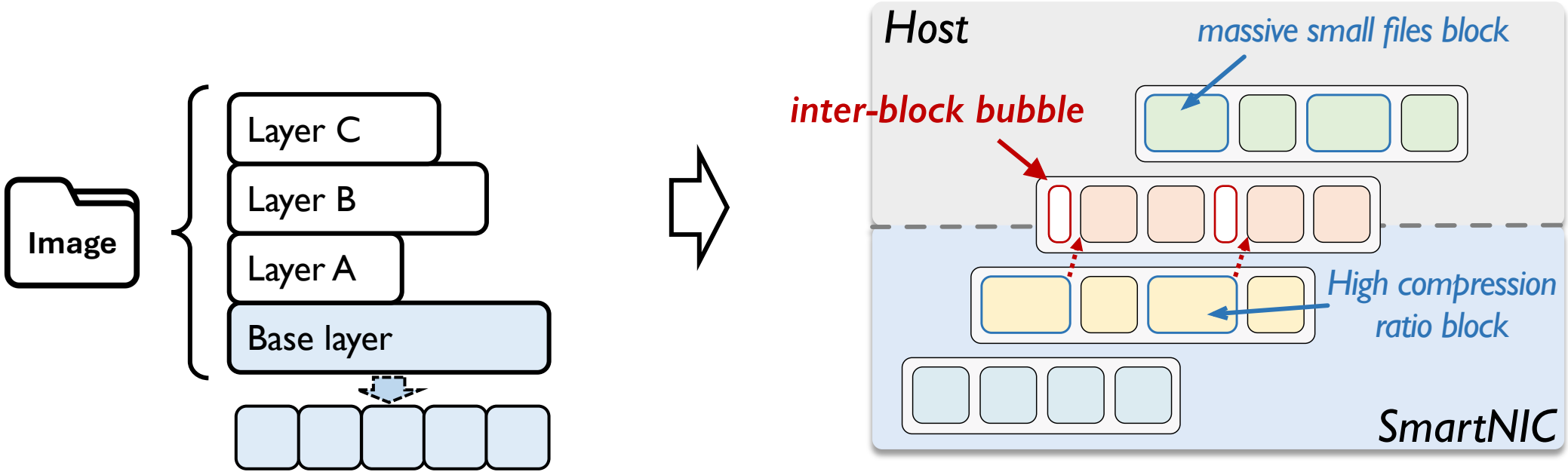
- ✔ **Pipeline the workflow to enhance *efficiency***
  - ✘ Layer-based pipelines ⇒ Limited parallelism & idle bubble
  - ✘ Time-based pipelines ⇒ Intra-cycle bubble



A fixed *cycle* has varying *processing capabilities* at different stages

Design II: Pipeline-based Data-driven Workflow

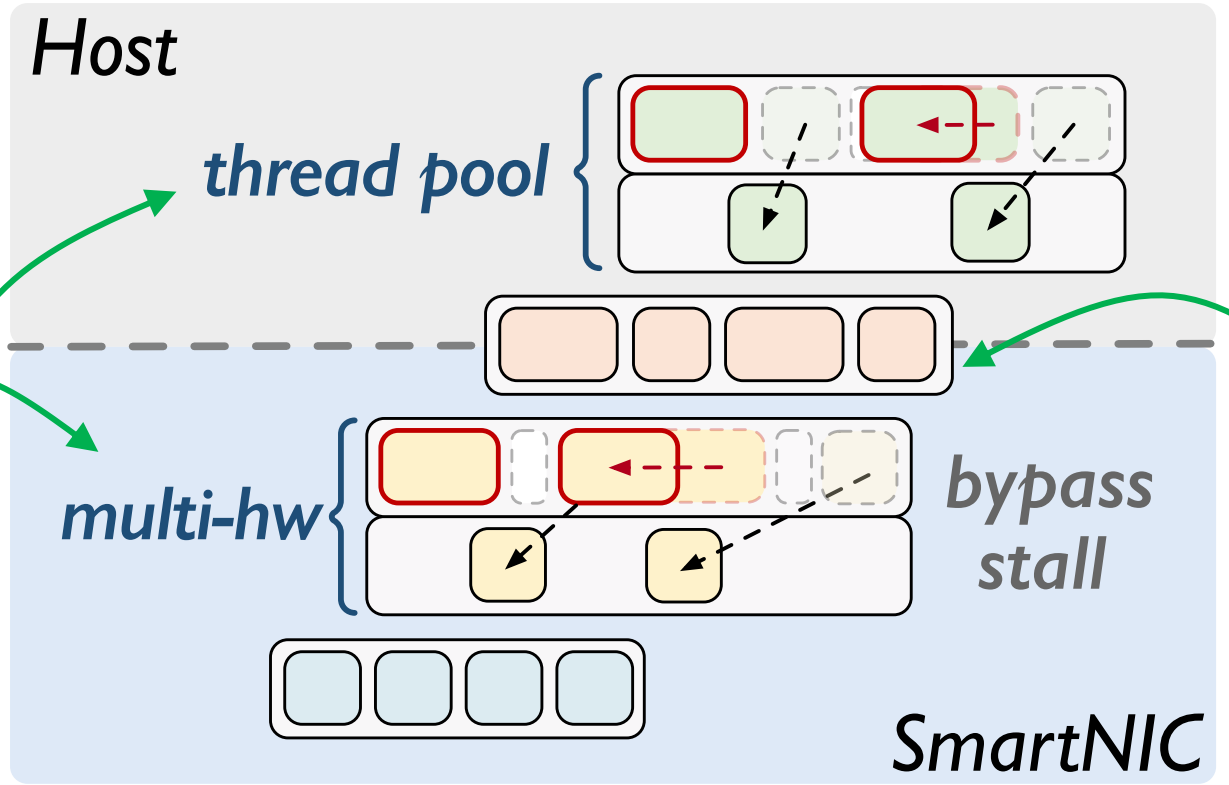
- ✔ **Pipeline the workflow to enhance *efficiency***
  - ✘ Layer-based pipelines ⇒ Limited parallelism & idle bubble
  - ✘ Time-based pipelines ⇒ Intra-cycle bubble
  - ✘ Block-based pipelines ⇒ Inter-block bubble



A fixed *block size* has varying *processing time* at different stages

Design II: Pipeline-based Data-driven Workflow

- **Pipeline the workflow to enhance *efficiency***
  - Block-based redundant pipelines

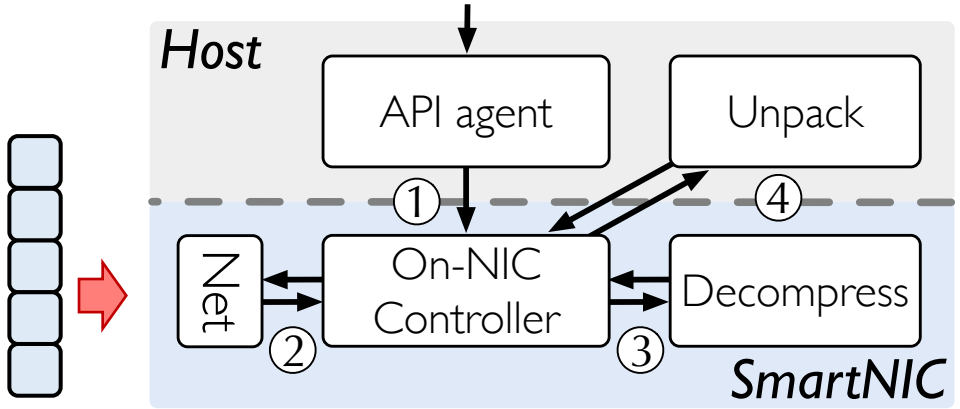


② Redundant hardware to address inter-block bubble

① Block-based pipeline to address intra-cycle bubble

Design II: Pipeline-based Data-driven Workflow

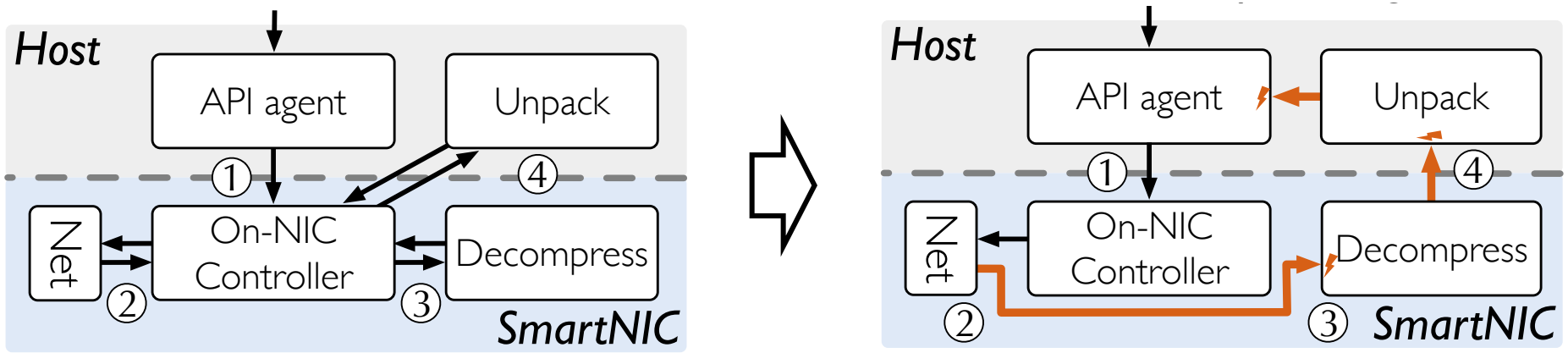
- ✔ **Data-driven workflow eliminates the overhead of *control messages***
  - ✘ Interact with all hardware components to orchestration pipelines
  - ✘ Block-based pipelines introduces massive interactions for each block



(a) Control-driven workflow.

Design II: Pipeline-based Data-driven Workflow

- **Data-driven workflow eliminates the overhead of *control messages***
  - Each block contains two pieces of state information
  - `block_index` specifies the position of a block in the layer and image
  - `block_status` indicates whether the block is compressed or not

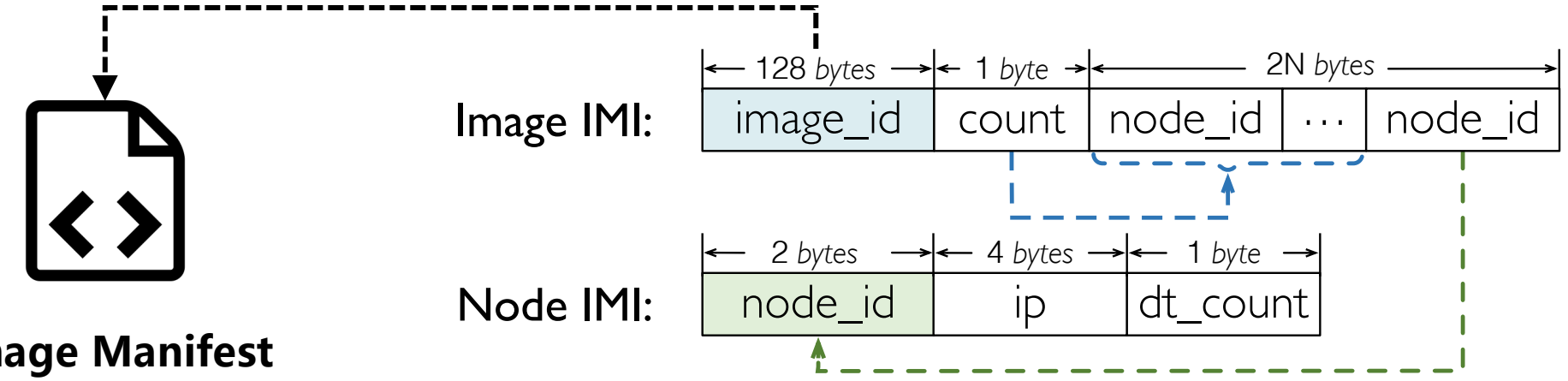


(a) Control-driven workflow.

(b) Data-driven workflow (`Poby`).

Design III: Distributed Image Download

- **Image metadata index (IMI) records *all images* and *node* info**
  - Track the available locations of images ⇒ *Image IMI*
  - Track the transmission workload of each node ⇒ *Node IMI*
  - Associate the image digest with the image manifest ⇒ *Image IMI*

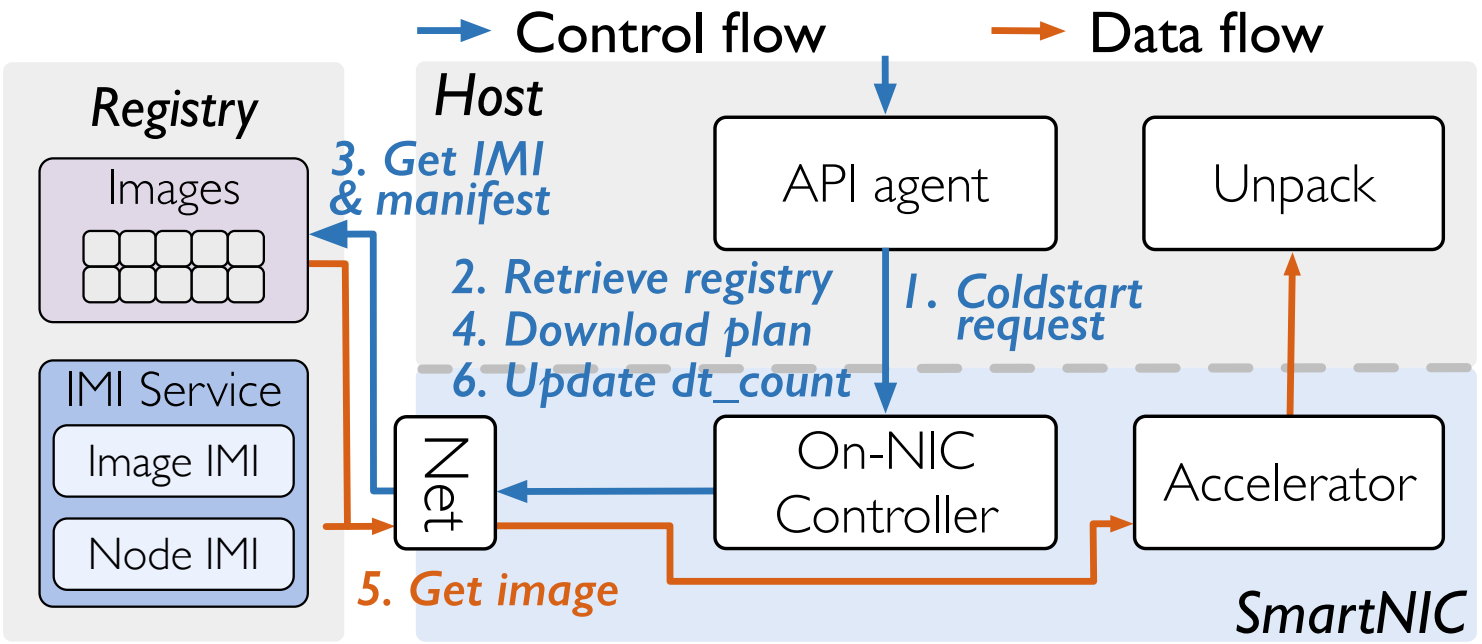


A system with ***50,000*** nodes and ***1,000,000*** images only require ***< 140 MB***

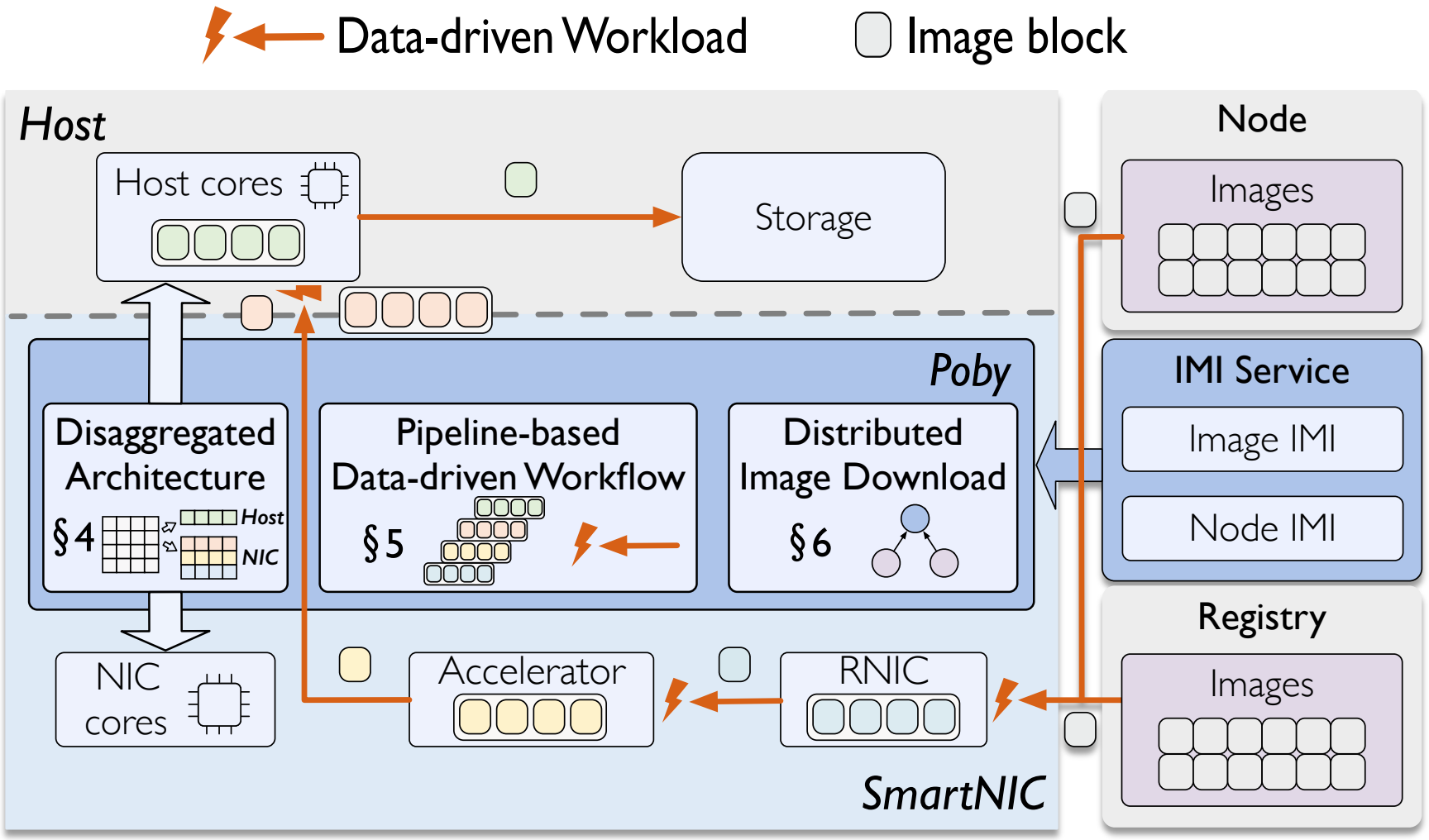
It is trivial to keep all IMIs in memory

Design III: Distributed Image Download

- **Best-effort distributed download incurs *no overhead***
  - Download from nodes that happen to contain the required image
  - Retains the original centralized download scheme as a fallback



Poby Overview



Please refer to our paper for more details

Evaluation

Experimental Setup

- Testbed
  - Prototype based on Bluefield-2
  - 2 Intel Xeon Gold 6226R servers with 256 GB memory
  - 1 server with dual-port 100Gbps ConnectX-5 NIC
- Baselines
  - containerd & iSulad
  - Kraken (Uber'18)
  - FaaSNET (ATC'21)

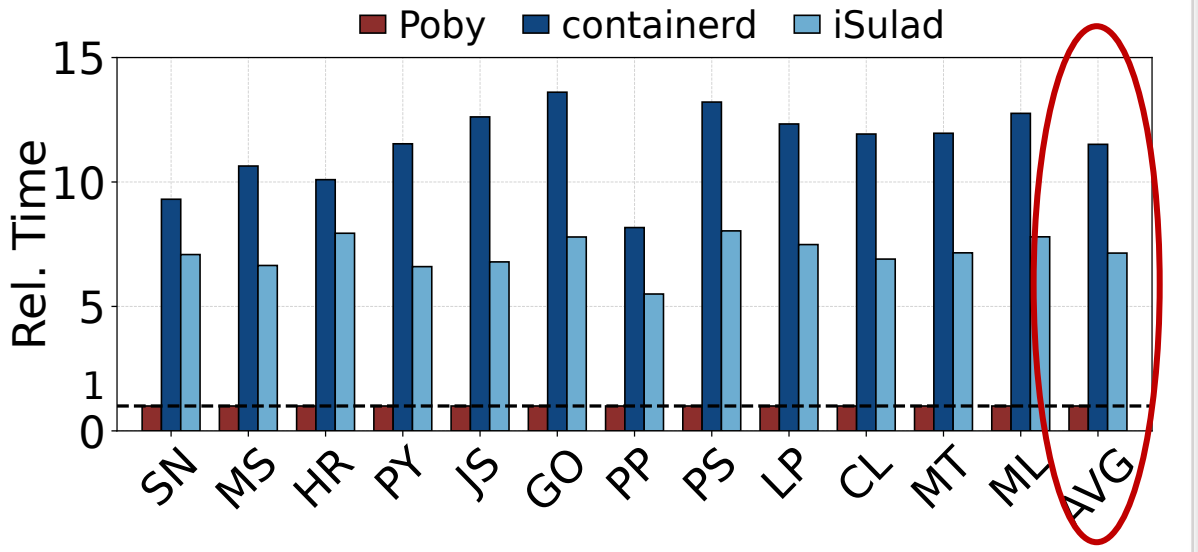
- Workloads

Table 1: Workload images adopted from benchmark suites.

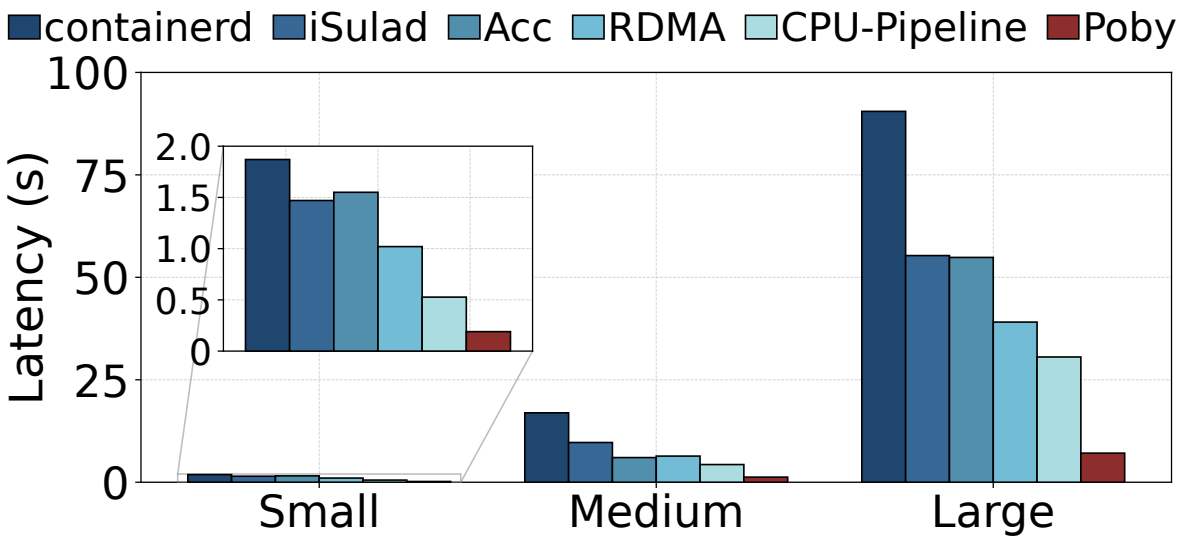
	Name	Specification	Source
Micro Service	Social Network (*SN*)	9 layers, 90.6MB	DeathStar Bench [24]
	Media Service (*MS*)	23 layers, 286.6MB	
	Hotel Reservation (*HR, small*)	18 layers, 39.6MB	
FaaS	Python (*PY*)	30 layers, 155.1MB	OpenWhisk [22]
	Node.js (*JS*)	21 layers, 356.3MB	
	Golang (*GO, medium*)	20 layers, 281.6MB	
	PHP (*PP*)	37 layers, 192.7MB	
	pyaes (PS)	33 layers, 169.9MB	Function Bench [30]
	linpack (*LP*)	33 layers, 214.6MB	
	chameleon (*CL*)	33 layers, 170.1MB	
	matmul (*MT*)	33 layers, 214.6MB	
	ML Model (*ML, large*)	37 layers, 2.45GB	

Performance Evaluation

E2E Performance Improvement



Performance Improvements Contributions



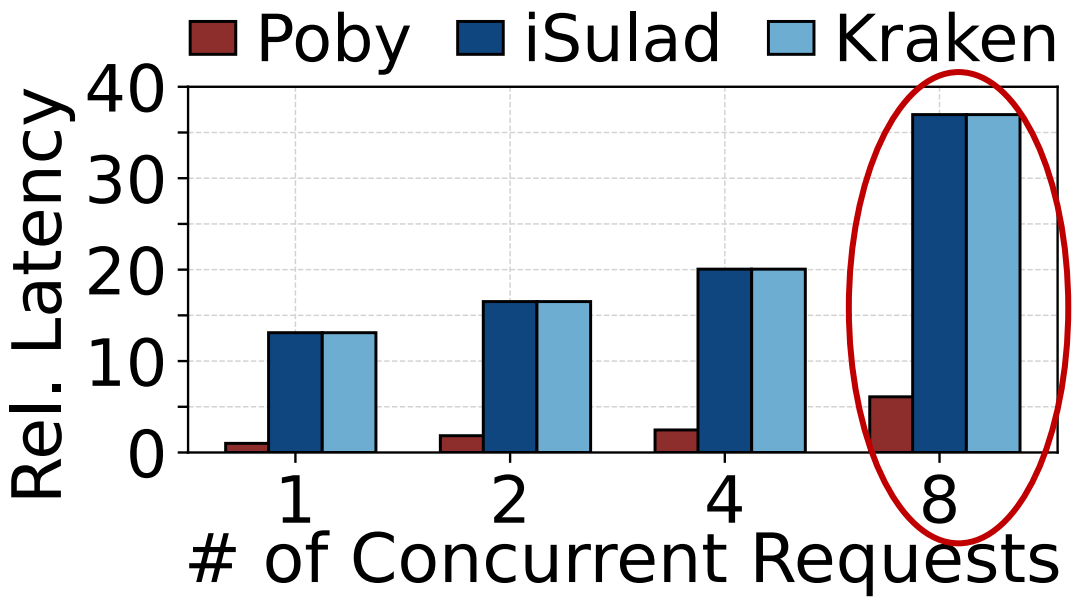
Up to ***11.5×*** and ***7.1×*** faster than baselines on average

HW (Acc & RDMA) reduces ***20%*** latency

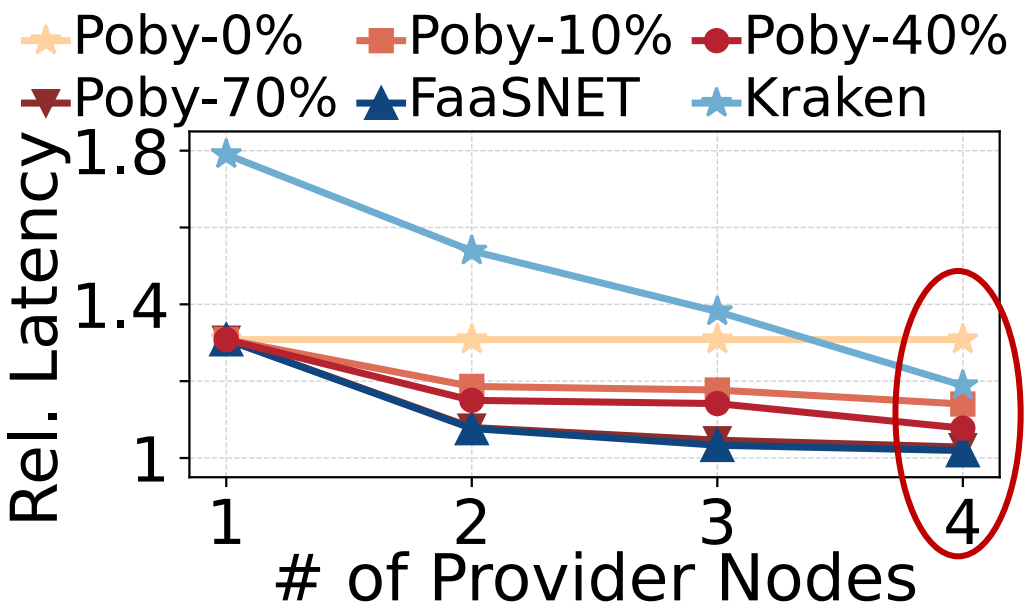
Pipeline (CPU-Pipeline) reduces ***34%*** latency

Concurrency and Scalability

Concurrency



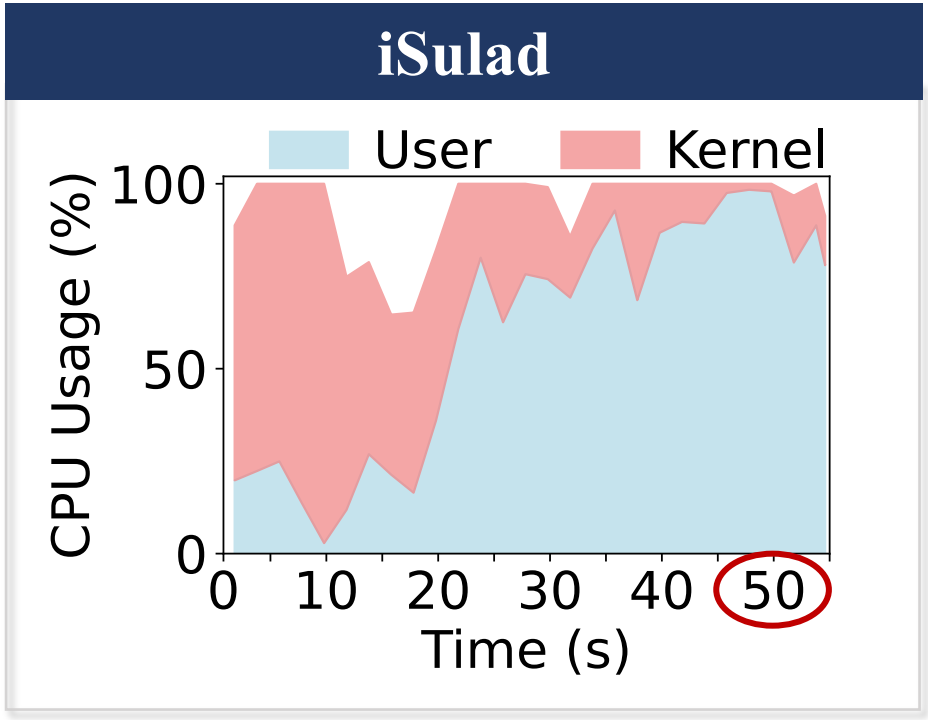
Scalability

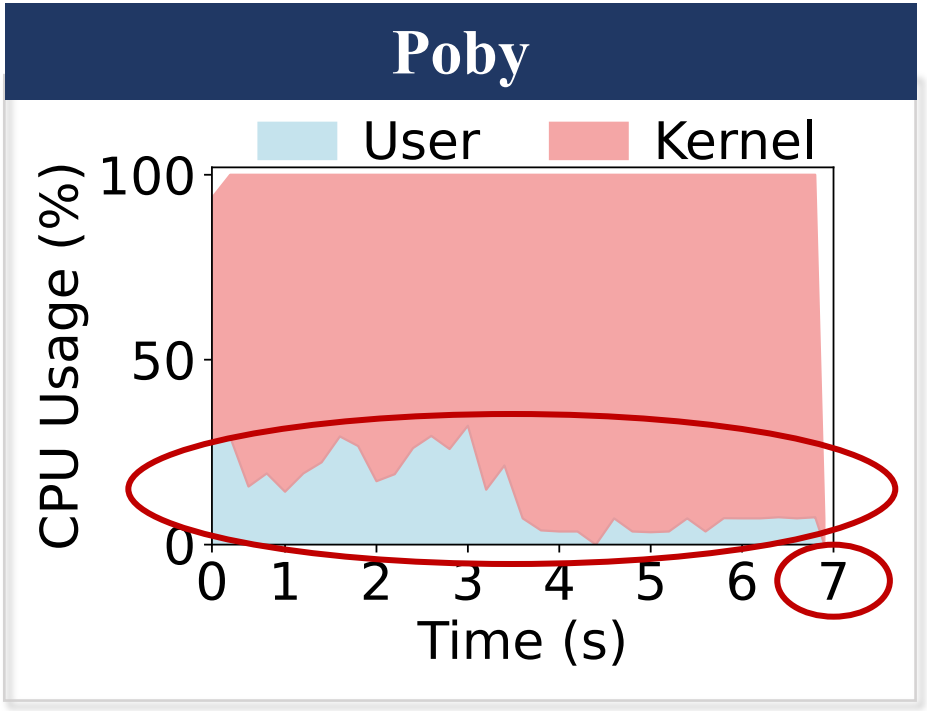


4.8× faster than baselines under 8 concurrent coldstart requests

Achieve comparable scalability to baselines with ***nearly no overhead***

Resource Demand





Offload ***45.3%*** of host side user-mode CPU usage to SmartNIC

Reduce ***87.5%*** of CPU usage time

Conclusion

- **Takeaways**
  - Coldstart from image provisioning is inevitable, SmartNIC can help fundamentally resolve the overlooked issue
- **Poby: software-hardware collaborative coldstart offload system**
  - Network-wide Disaggregated Architecture
  - Pipeline-based Data-driven Workflow
  - Distributed Image Download
- **Outperform the industry baseline containerd by *13.2×***

Thanks!
Q&A

changzihao95@gmail.com