On the Impact of Isolation Costs on Locality-aware Cloud Scheduling

Ankit Bhardwaj, **Meghana Gupta**, Ryan Stutsman

University of Utah

THE UNIVERSITY OF UTAH®

Scalable Computer Systems Lab
www.utah.systems

Code Isolation-cost Aware Scheduling

Cloud Networking Performance → 100 Gbps, microsecond round-trips

Rethink of code isolation schemes → Meltdown, Spectre, VT-x, eBPF, WASM

Granular, Serverless Applications → Visibility and Placement a fine grain

Three recent shifts in the cloud

Code Isolation-cost Aware Scheduling

Cloud Networking Performance → 100 Gbps, microsecond round-trips

Rethink of code isolation schemes → Meltdown, Spectre, VT-x, eBPF, WASM

Granular, Serverless Applications → Visibility and Placement a fine grain

Three recent shifts in the cloud

Code Isolation-cost Aware Scheduling

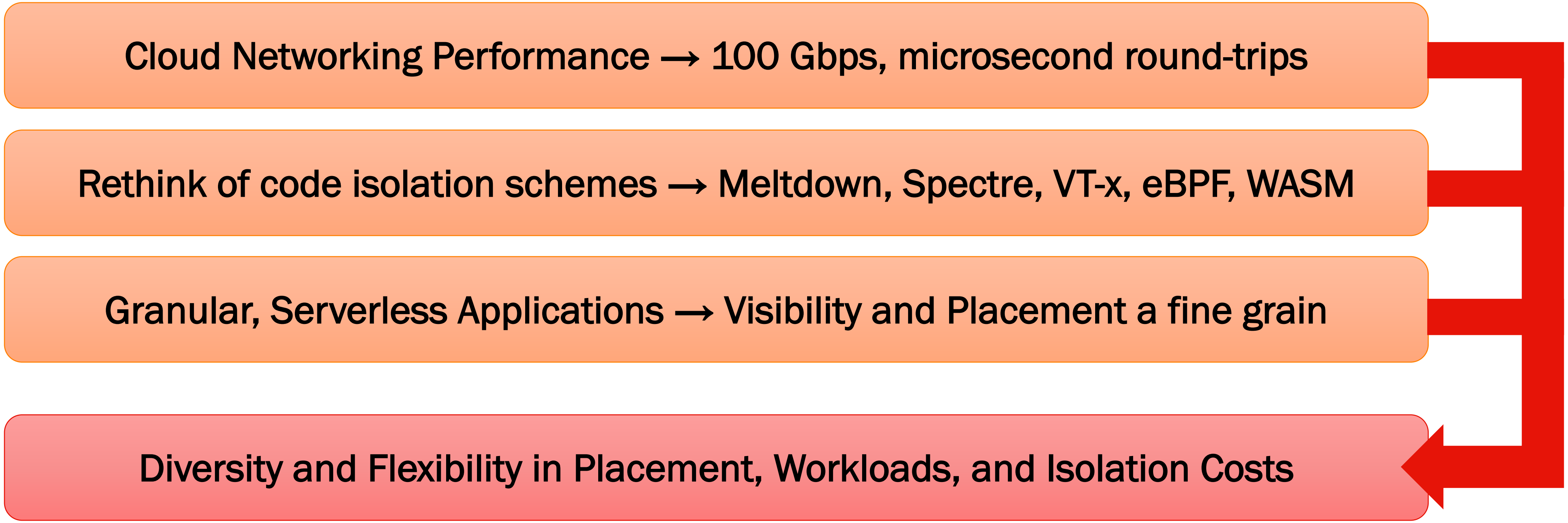
Cloud Networking Performance → 100 Gbps, microsecond round-trips

Rethink of code isolation schemes → Meltdown, Spectre, VT-x, eBPF, WASM

Granular, Serverless Applications → Visibility and Placement a fine grain

Three recent shifts in the cloud

Code Isolation-cost Aware Scheduling



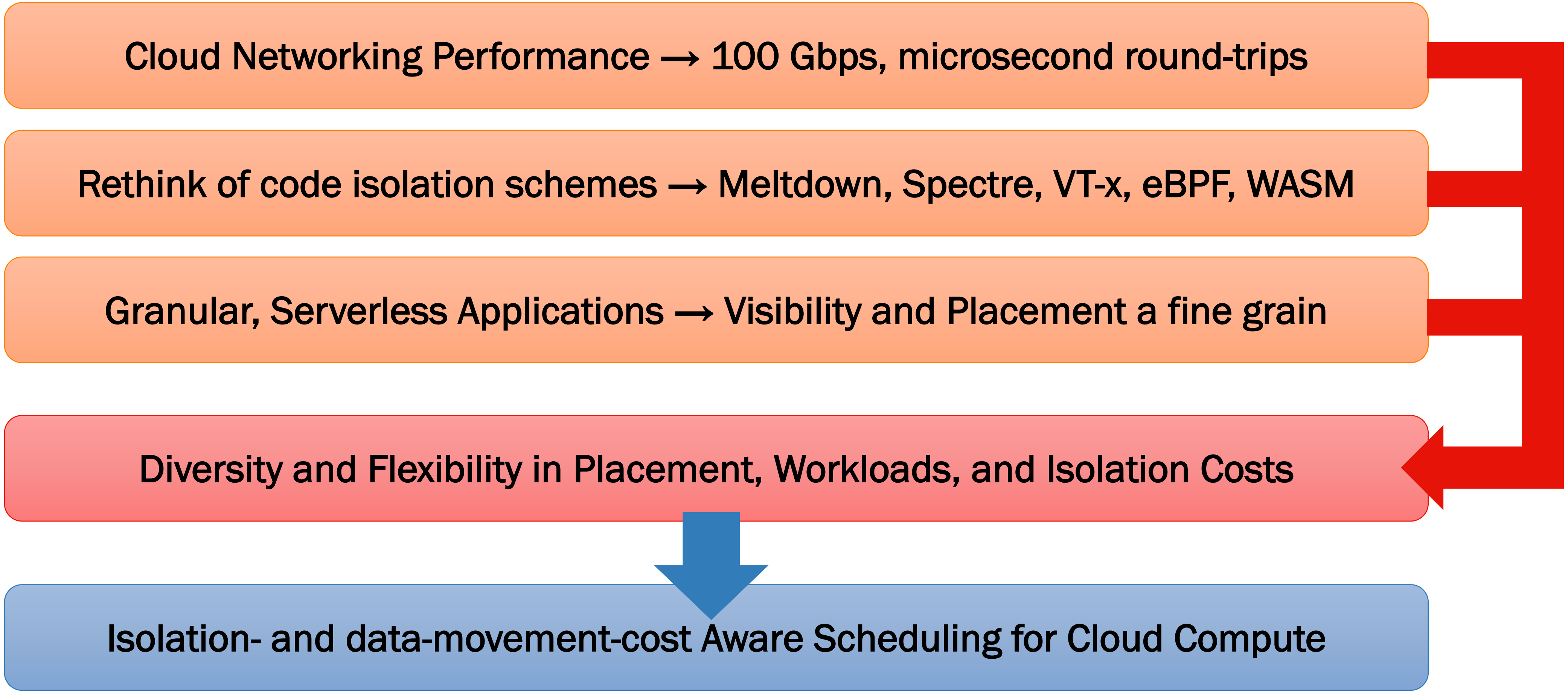
Cloud Networking Performance → 100 Gbps, microsecond round-trips

Rethink of code isolation schemes → Meltdown, Spectre, VT-x, eBPF, WASM

Granular, Serverless Applications → Visibility and Placement a fine grain

Diversity and Flexibility in Placement, Workloads, and Isolation Costs

Code Isolation-cost Aware Scheduling



Cloud Networking Performance → 100 Gbps, microsecond round-trips

Rethink of code isolation schemes → Meltdown, Spectre, VT-x, eBPF, WASM

Granular, Serverless Applications → Visibility and Placement a fine grain

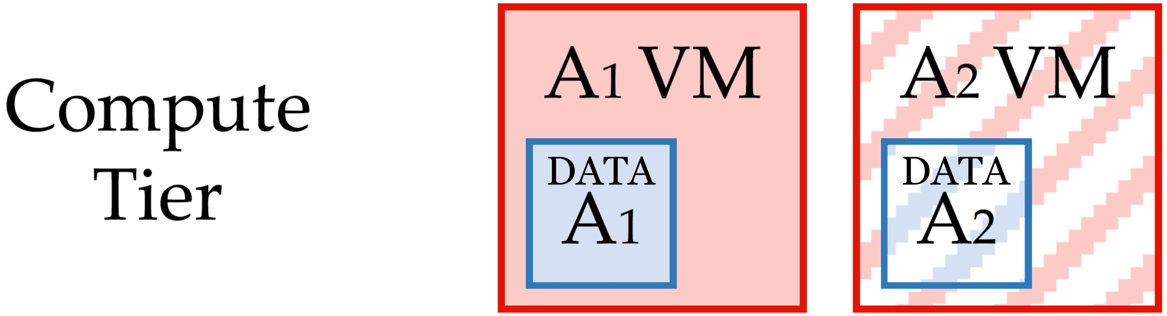
Diversity and Flexibility in Placement, Workloads, and Isolation Costs

Isolation- and data-movement-cost Aware Scheduling for Cloud Compute

It is time for a holistic, cost-aware approach to scheduling in the cloud

Past: State + Application on One VM

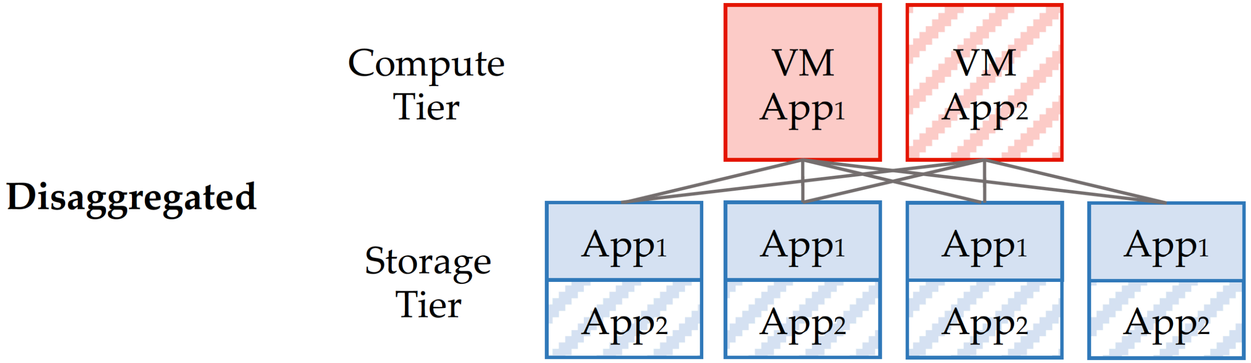
- Compute/storage together on one machine; VMs access state locally



- Problem: Resource stranding
  - Idle compute when storage capacity is the limiting factor
  - Idle storage when compute capacity is the limiting factor
  - Costly to reorganize

Today: Disaggregation

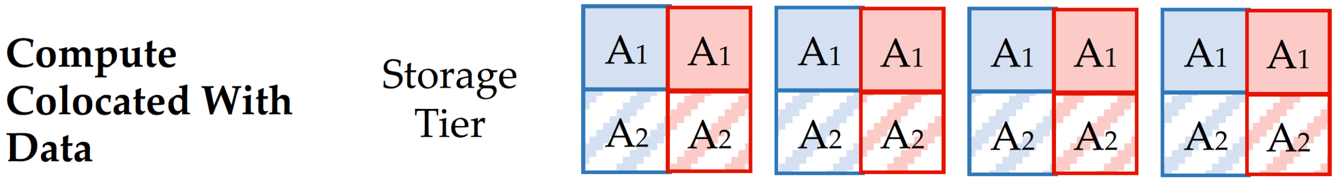
- Solution: Separate compute from storage



- New Problem: High data movement costs (multiple gets/puts)
  - RPC, serialization/deserialization
  - TCP/transport
  - memcpys
  - Substantial costs at gigabits/second

Move compute to storage at finer grain?

- Solution: storage-side computation over stored data



- But, high tenant density at storage to homogenize/balance load
- Need granular decomposition of application logic

- Problem: Many tenants sharing storage; code isolation is hard
  - Process creation and context switch add up

Key Idea: Isolation-cost Aware Scheduler

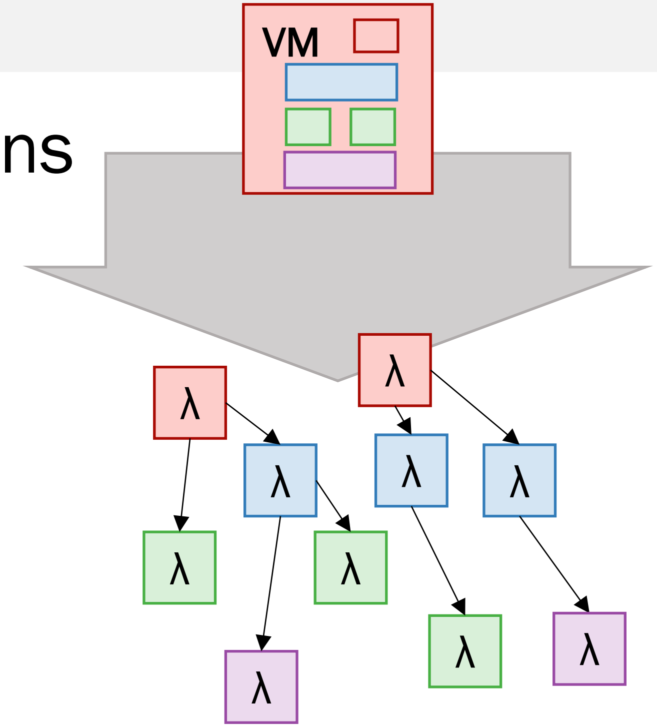
- Placement of computation in the cloud can improve efficiency
  - by eliminating data movement,
  - but it also must reason about code isolation costs to do so.
- Profile
  - inter-function interaction in applications,
  - data access and locality patterns,
  - networking, dispatch, and isolation domain context switch costs
- Global fine-grained, core-level choices at microsecond-timescales

Challenges for Isolation-cost Aware Scheduling

- Need for Fine-grained Applications
- Workload Characterization
- Profiling and Understanding Context Switch Costs
- Provisioning, Re-provisioning, and Placement
- Dealing with Intermediate State

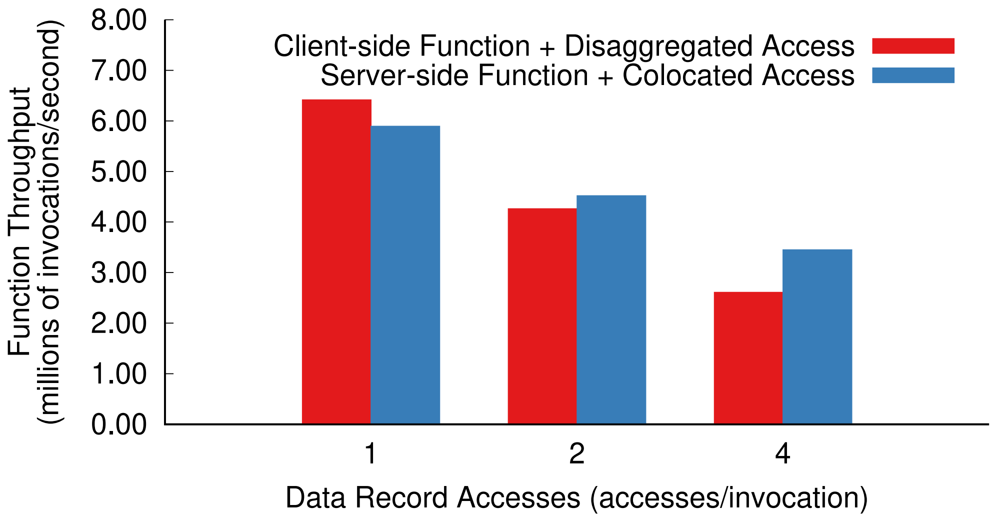
Challenge #0: Need Finer-grained Apps

- Scheduler must be able to "see" into applications to optimize
- **Solution**: serverless
  - Functions can be individually placed
  - Creates visibility into applications
  - Supports alternative isolation schemes
  - Malleable interface
- Today implementations do not tap into these potential benefits



Challenge #1: Workload Characterization

- Problem: No insight into function's network and data access costs
- Solution: Profile functions to capture
  - data access patterns and locality
  - runtime distribution



- Place functions that access many records or much data at storage
- Dynamically shift to idle compute when server is overloaded
- Even simple schemes can work: counting accesses & runtime

Challenge #2: Code Isolation Costs

Problem: isolation costs vary depending on workload

VMs: hw protection & dispatch

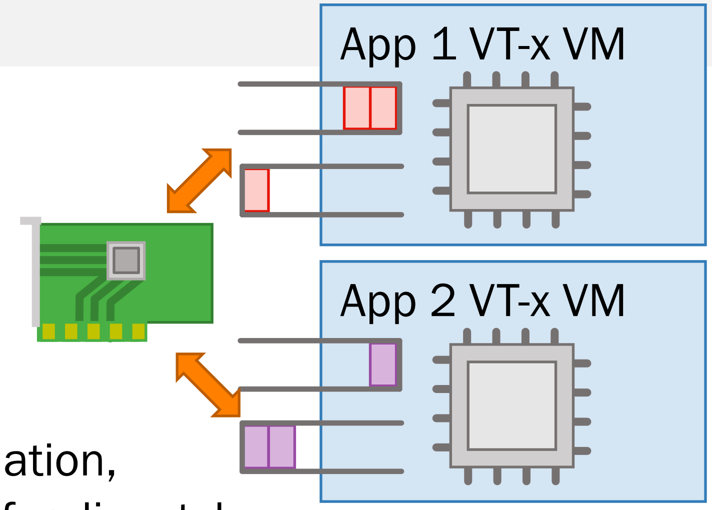
- Too expensive to context switch
- Good if high per-tenant throughput

Containers: sw dispatch

- Need ms-scale length requests
- Good for timesharing CPU

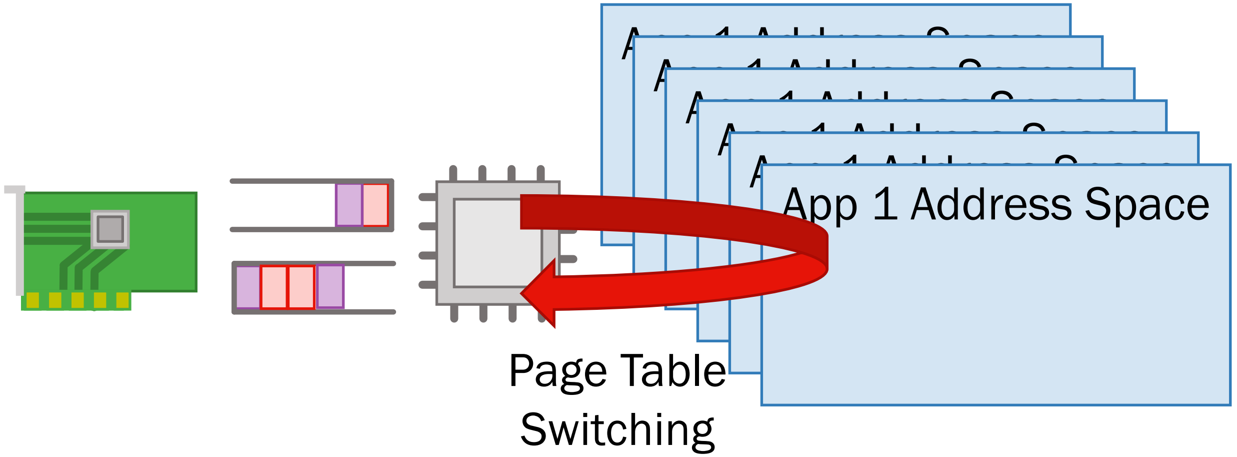
Language Runtimes: pure sw

- Good for short-running functions with constrained logic

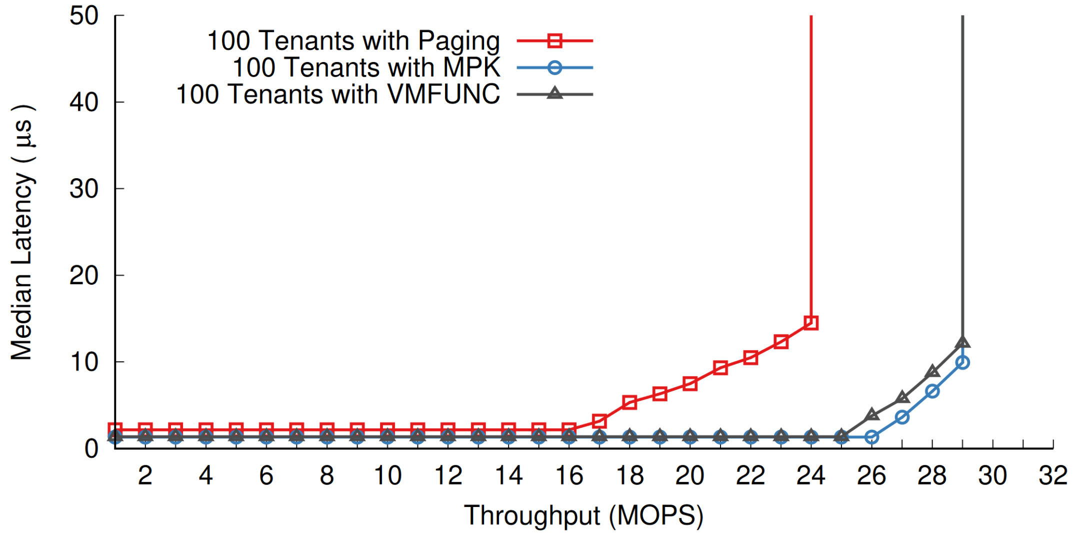


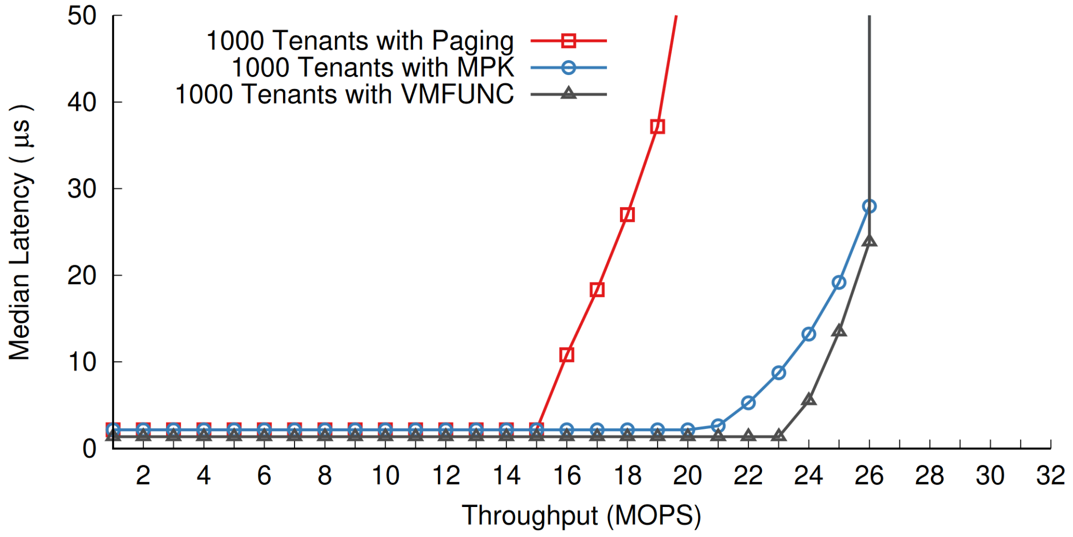
VT-x VMs for isolation,
SR-IOV+IOMMU for dispatch

Processes for isolation,
software demultiplexing for dispatch



Comparing Three Hw Isolation Schemes





- Paging/conventional process context switch is always costly
- Low tenant counts → MPK Page Table Entry Coloring Fastest
- Higher tenants counts → Extended Page Table Switching Fastest

Best scheme depends on tenant count and request rates

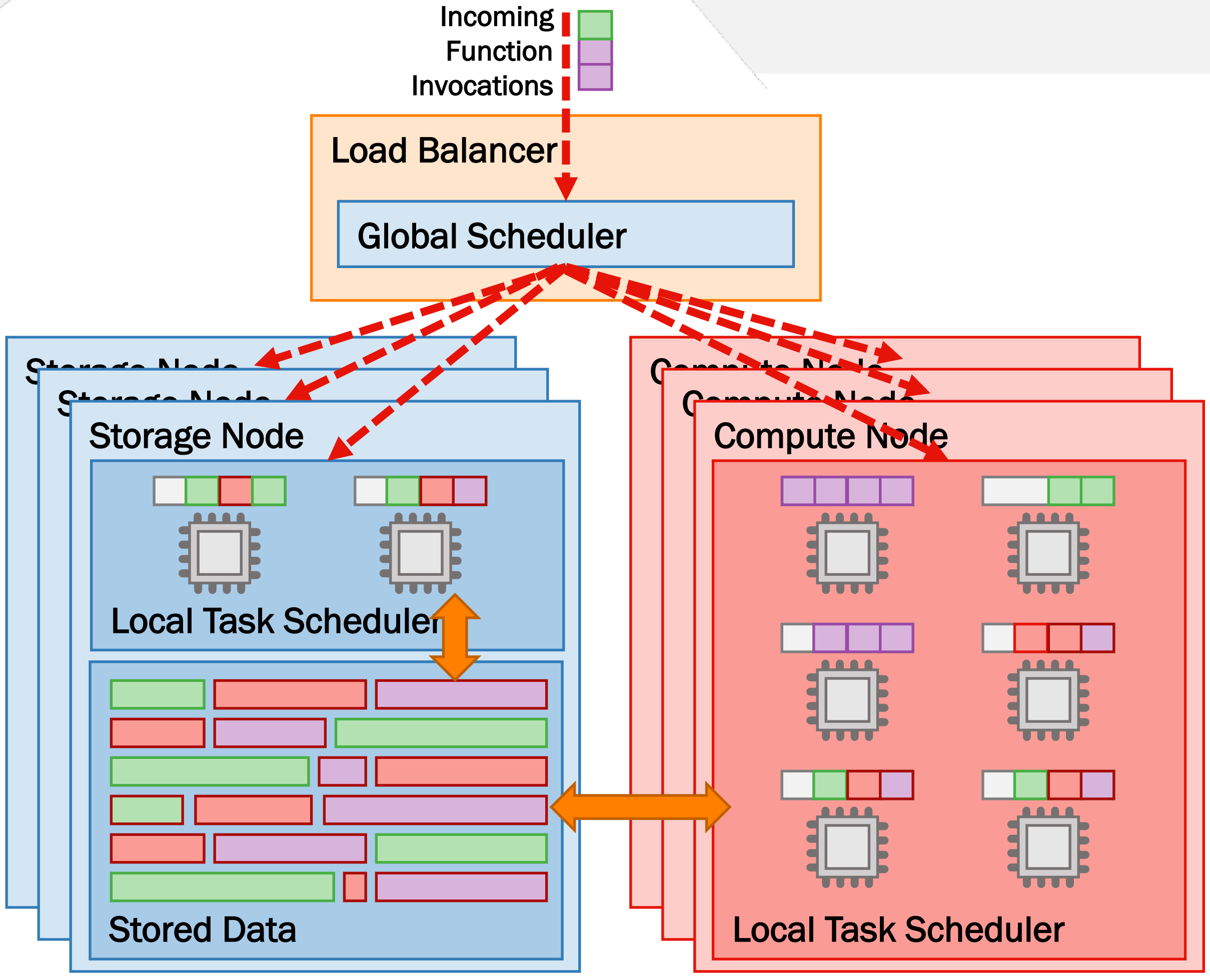
Challenge #3: Provisioning & Placement

- Problem: Function properties change over time
  - in data access patterns
  - in computational costs
  - in distribution of functions invoked

- Churn and instability forces new placement decisions
- VMs, containers, etc have different start, stop, migration costs

- Solution: scheduling must model stability and variance of workload
  - In compute costs, invocation frequency, and data access

Preliminary Design Ideas



Task Dispatching

- Two-level scheduling avoids idle CPUs but limits queue imbalance
- History at global level, route invocations to avoid context switching
- Global knowledge of data placement

Statistics, Load, & Prediction

- Core and task level stats collection
    - Push via RDMA writes
    - Low-cost with frequent updates
    - 100s to 1000s of machines pushing updates each second
- Use in assessing workload stability
    - Used by scheduler to promote/demote functions between isolation schemes

Discussion Questions

- Cloud process model
  - Cloud function interfaces (that differ from POSIX) are likely to take hold?
- Security risks
  - Larger attack surface, but works around vulnerabilities with less reengineering
  - Which isolation schemes and runtimes likely to be sufficiently trustworthy?
- Workloads
  - What will future, more granular serverless workloads look like?
  - What ways might there be to approximate these workloads using public data?
- Pricing
  - How might improved but hard-to-predict efficiency gains be reflected in pricing?

Conclusion

- Kernel-bypass ⟶ low-latency, high-throughput storage services
  - These gains are now showing up in the cloud
  - Fast networks ⟶ more data movement

- Small functions over data, but code isolation cuts into gains

- Key idea: different code isolation schemes have different costs
  - Dynamically understand data movement and code isolation costs
  - Run different functions with different schemes based on runtime profiling

- For more details, check out our project website or reach out to me at meghana@cs.utah.edu.