Forsythia: Computing Over Hundreds of Thousands of Potentially Spotty Machines

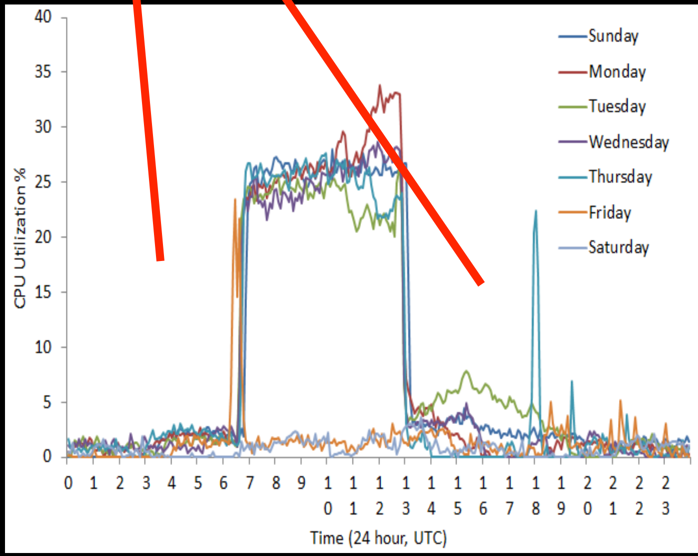
Zhihao Jia

Joint work with Dennis Fetterly, JP Martin, and Yuan Yu

Motivations

- Data center is hugely under utilized
  - Large numbers of idle cycles
  - Unlikely to be significantly improved very soon
  - Better to exploit transient idle machines
  - Amazon exposes spot instance
- Data-parallel applications needs increasingly more computing resources
  - E.g., Bing’s AIM project requires more than tens of thousands of CPU cores

Idle cycles



Server CPU utilization in Amazon EC2 cloud

http://huanliu.wordpress.com/2012/02/17/host-server-cpu-utilization-in-amazon-ec2-cloud/

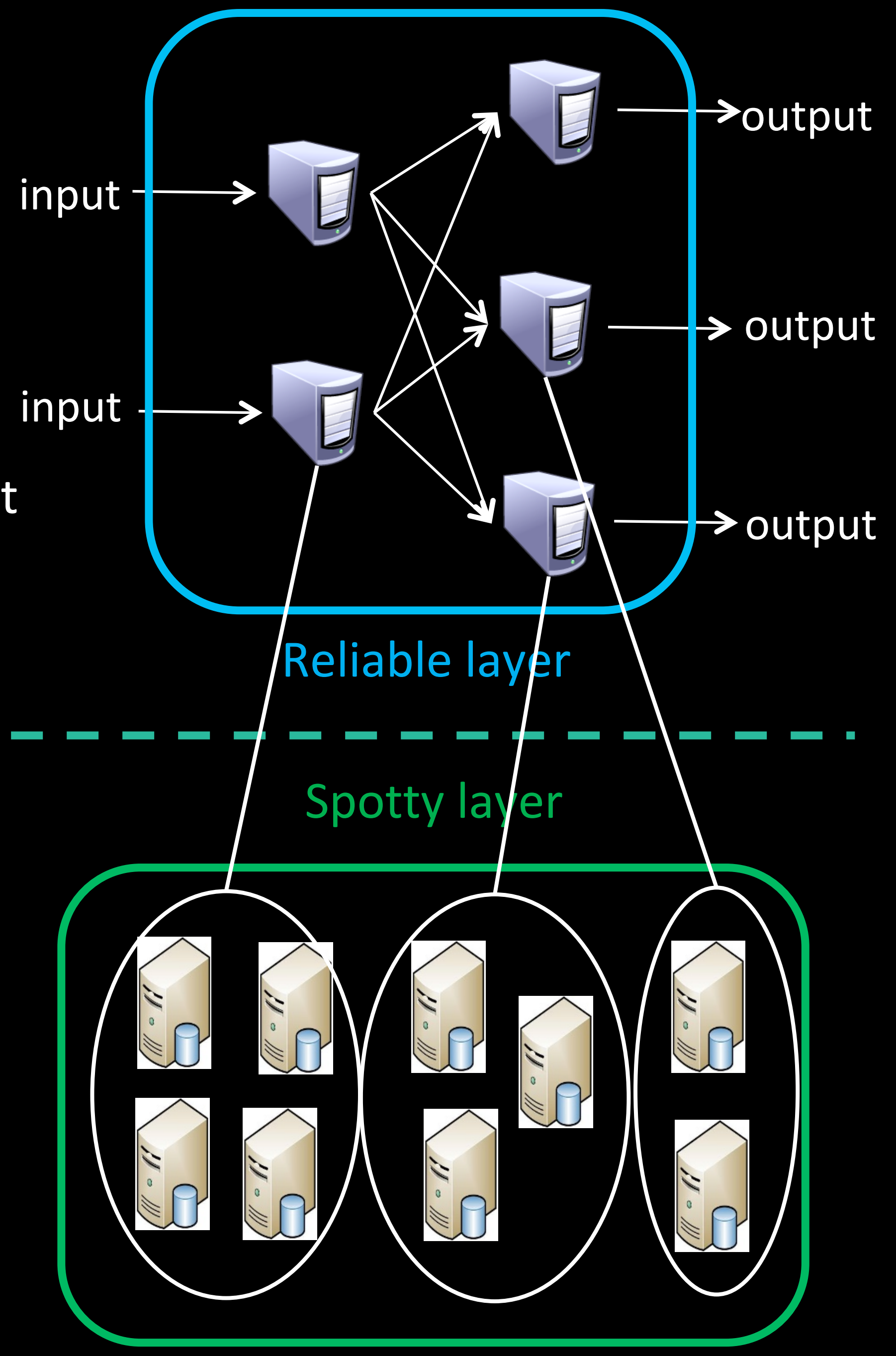
Is it possible to scale out applications by exploiting large numbers of transient machines?

Two Key Ideas

1. Break computation into standalone work items
    - Can be reassigned and reexecuted at anytime

2. Divide execution into two layers

- Reliable layer: support Dryad-like dataflow computation, further partition vertices into work items
    - E.g., machines in Dryad/Hadoop clusters
- Spotty layer: opportunistically execute work items
    - E.g., idle VMs in Azure
    - E.g., Amazon spot instances



Programming Abstraction

- Similar to threadpool
  - Virtual core: transient physical CPU core
    - Pre-emptible, unreliable, and unpredictable
  - Virtual core pool (VCPool): manage collection of virtual cores
- Easy to use

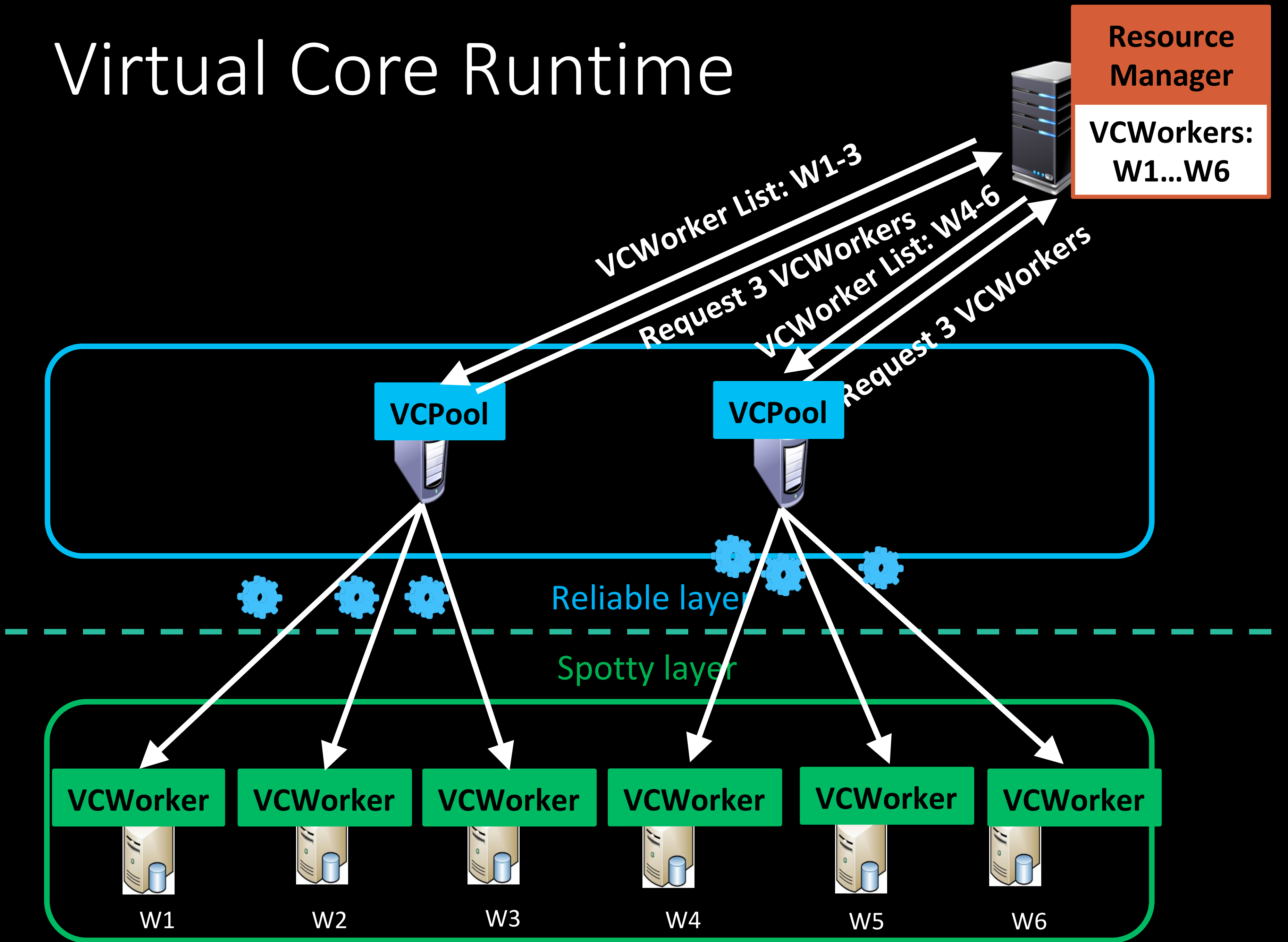
Step 1: Create a VCPool

Step 2: Enqueue work item into VCPool

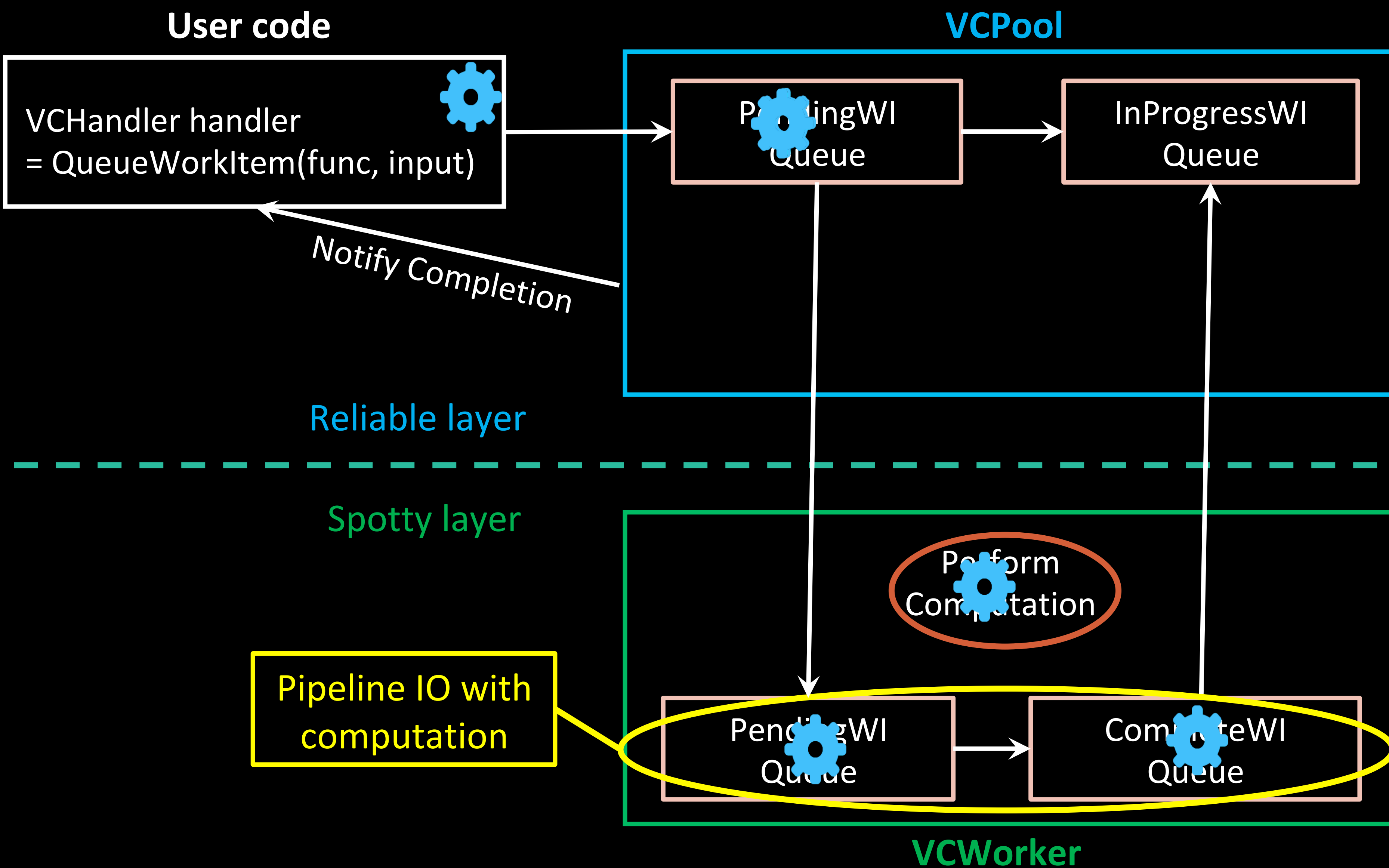
```
/* Example: compute func(input) on virtual core*/
VCPool pool = new VCPool(config);
VCHandler<R> handler = pool.QueueWorkItem<T, R> (func, input);
handler.WaitUntilDone();
/* access result via handler.result */
```

Step 3: Wait until the result is ready

Virtual Core Runtime



VCPool/VCWorker Architecture

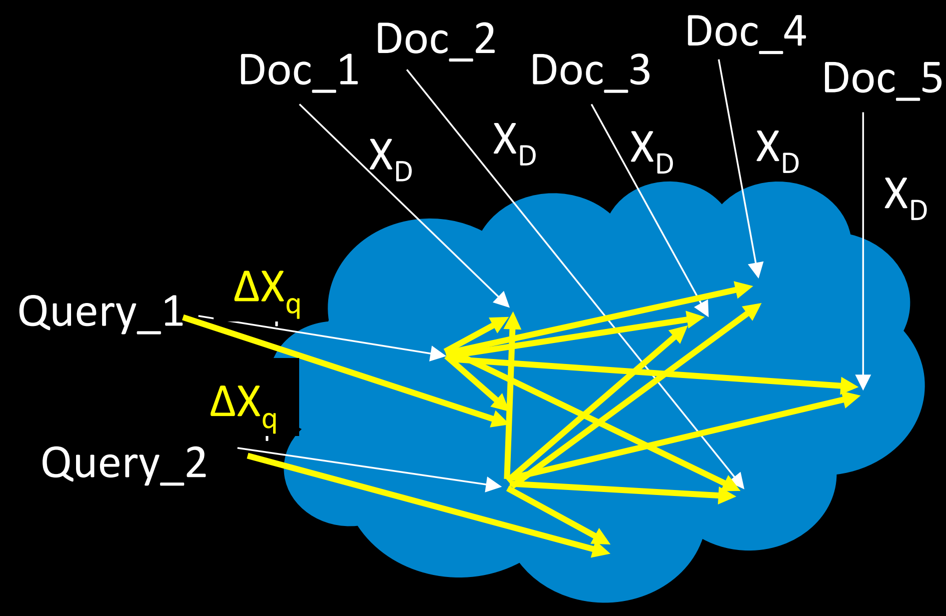


Bing AIM project: machine learning application for web search engine

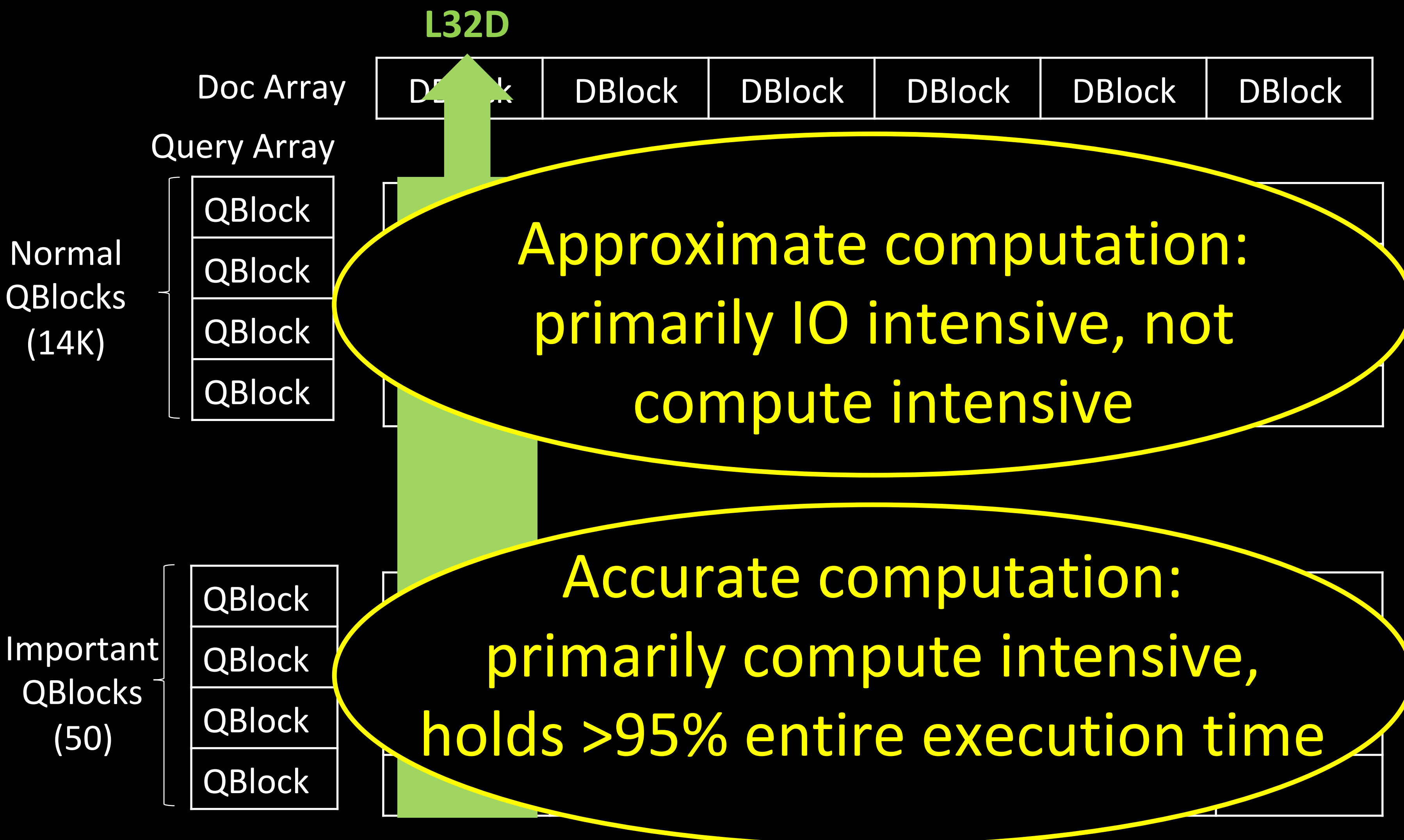
- Goal: mapping queries and docs to a high dimensional space, so that semantically-related docs and queries are nearby in terms of distance

Computations:

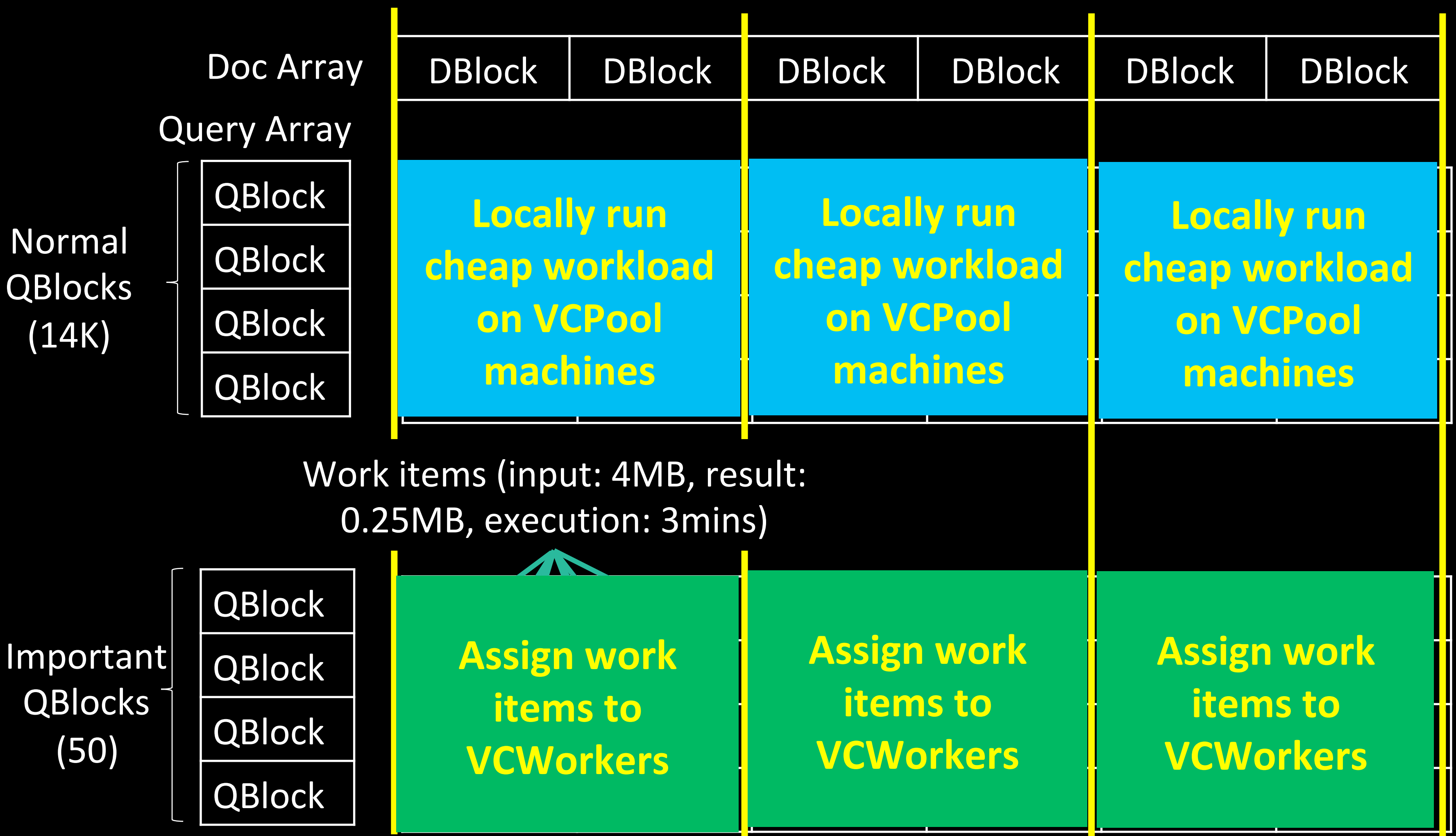
1. Queries/docs correlation
   - L32D
2. Vector gradient
   - Query updates



L32D: Correlations between Docs and Queries



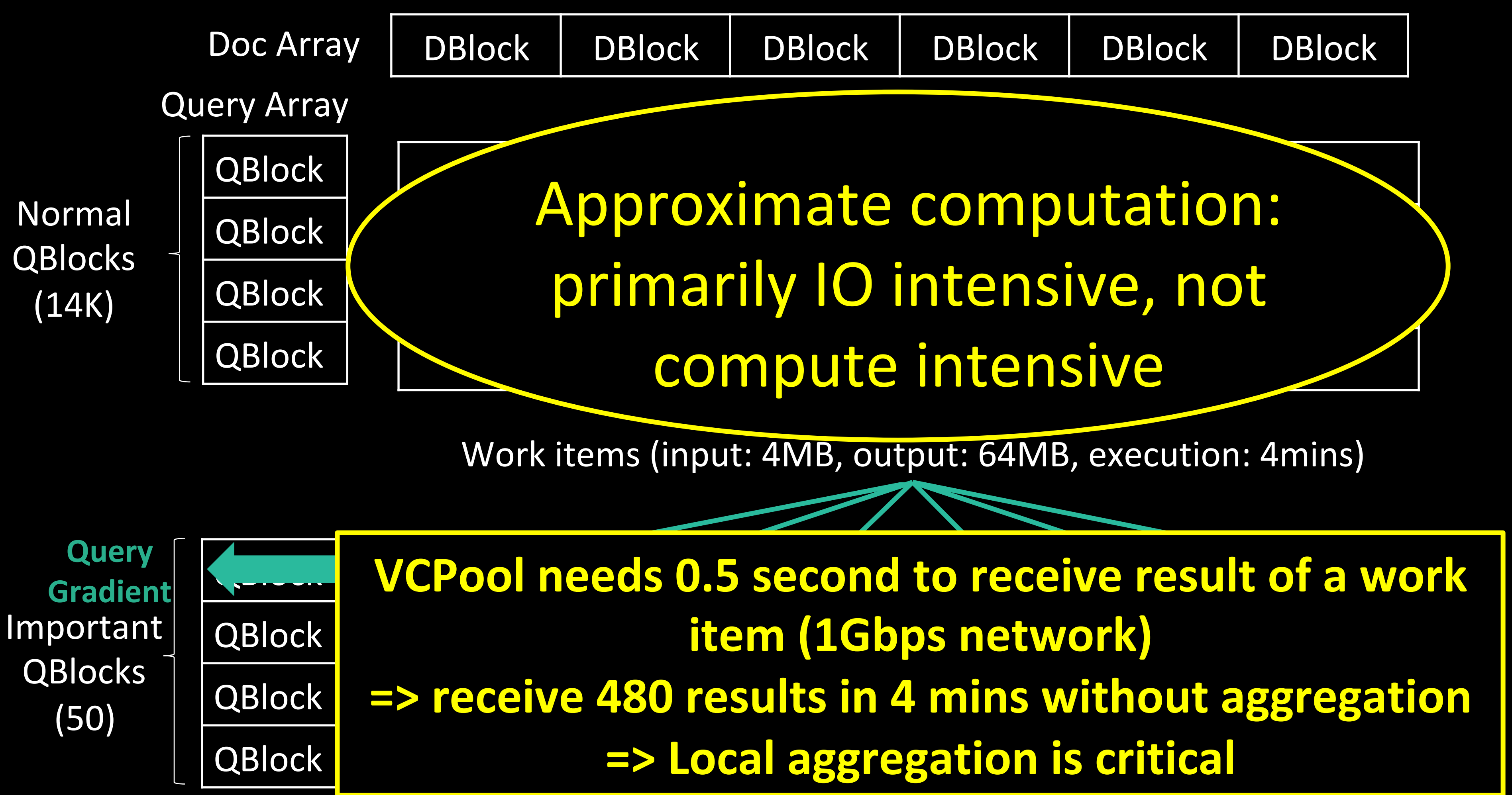
L32D: Virtual Core Design



L32D: Optimizations

- Scalability is bottlenecked by network bandwidth on reliable layer
- Optimization 1: Cache frequent input on VCWorkers
  - E.g., Cache all important QBlocks (50) on VCWorkers (~110MB in largest dataset) => shrink amortized input size
- Optimization 2: Allow local aggregation on VCWorkers
  - E.g., Final L32D result is the sum of all work items’ results.
  - Locally aggregate L32D result on VCWorkers => shrink amortized result size
- AIM result: VCPool network usage is reduced by a factor of 10, with both optimizations
- Generally useful for compute intensive applications

Query Update: Compute Vector Gradient for Query Mapping



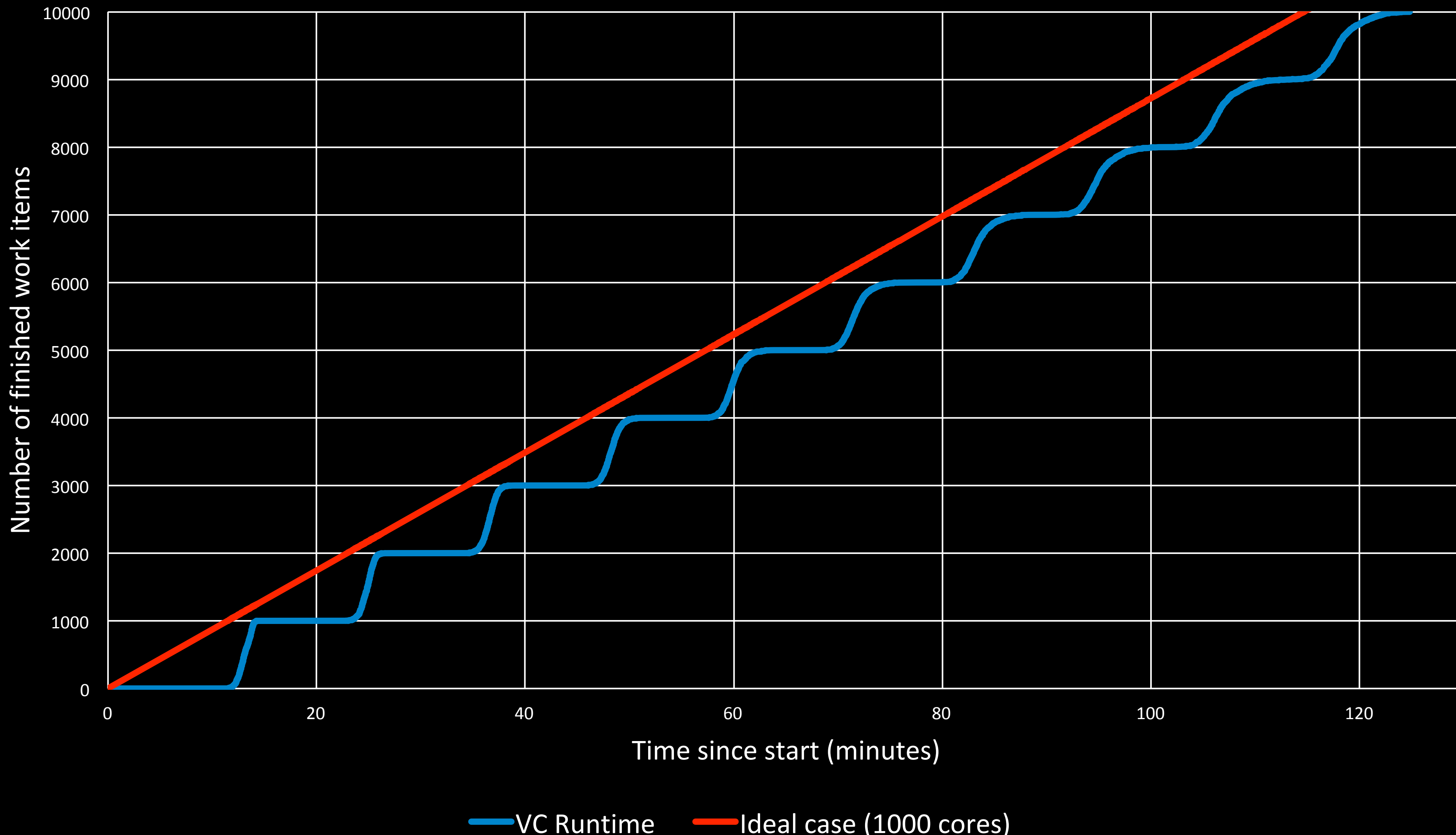
Evaluation

- Can single VCPool scale to thousands of VCWorkers?
- How would VCPool perform in presence of failure?
- Can multiple VCPools scale simultaneously?

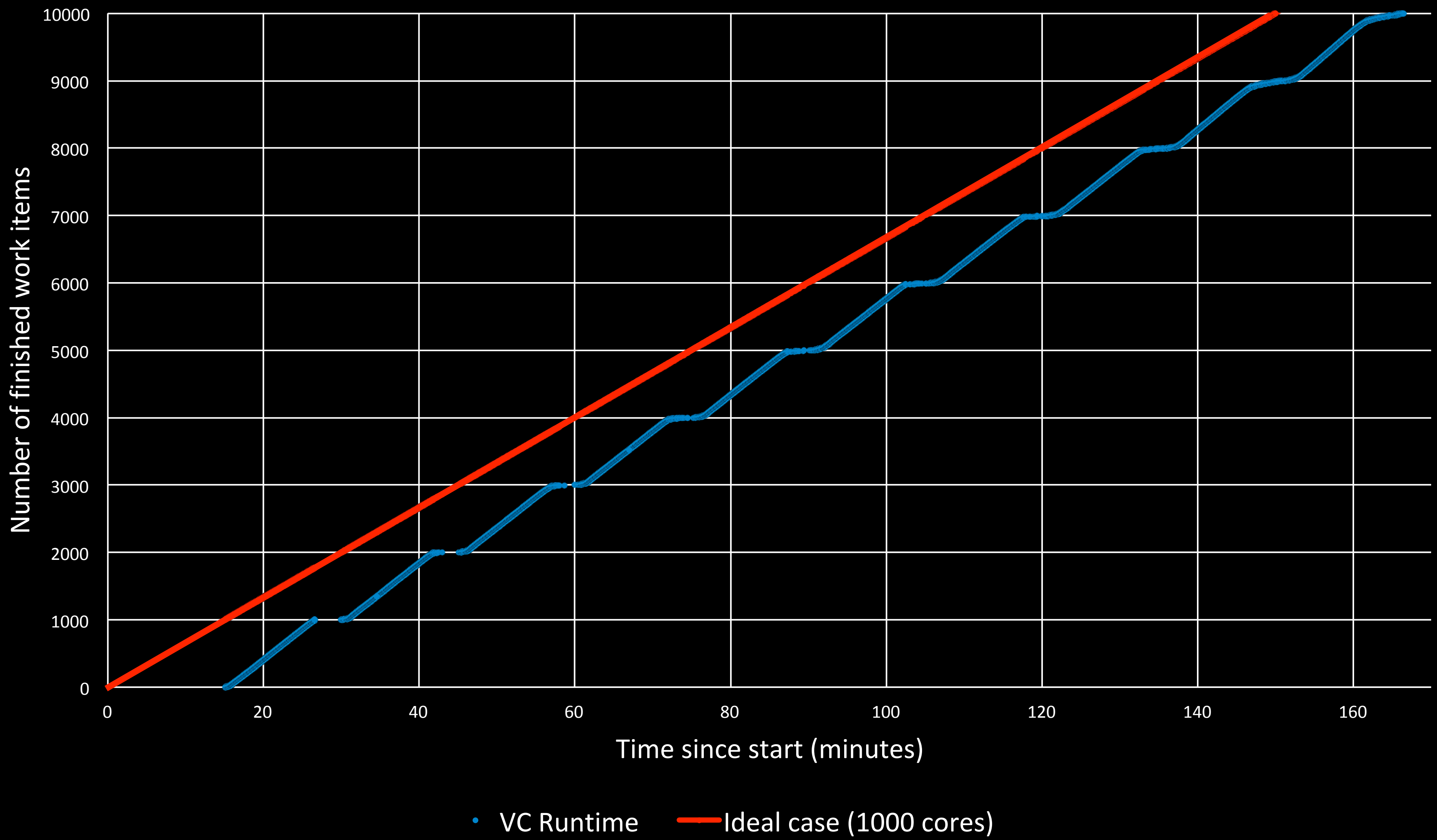
1. Can Single VCPool Scale to Thousands of VCWorkers?

- Birnam cluster: 8 cores on each machine, 1Gb network bandwidth
- Use AIM as the benchmark
- Start 1,000 VCWorkers on 125 Birnam machines
- Start a single VCPool that tries to manage all 1,000 VCWorkers
- Use both optimizations (caching frequent input, local aggregation)

L32D Throughput



Query Update Throughput



2. How would VCPool perform in presence of failure?

- Introduce failure model:
  - Divide time line into equivalent time slots, with length T
  - During a time slot, a VCWorker may stop working/ become unavailable with probability P
  - VCWorker independently makes decision during each time slot
- Experiments:
  - Single VCPool manages all 1,000 VCWorkers
  - T = 6mins, P = 20%

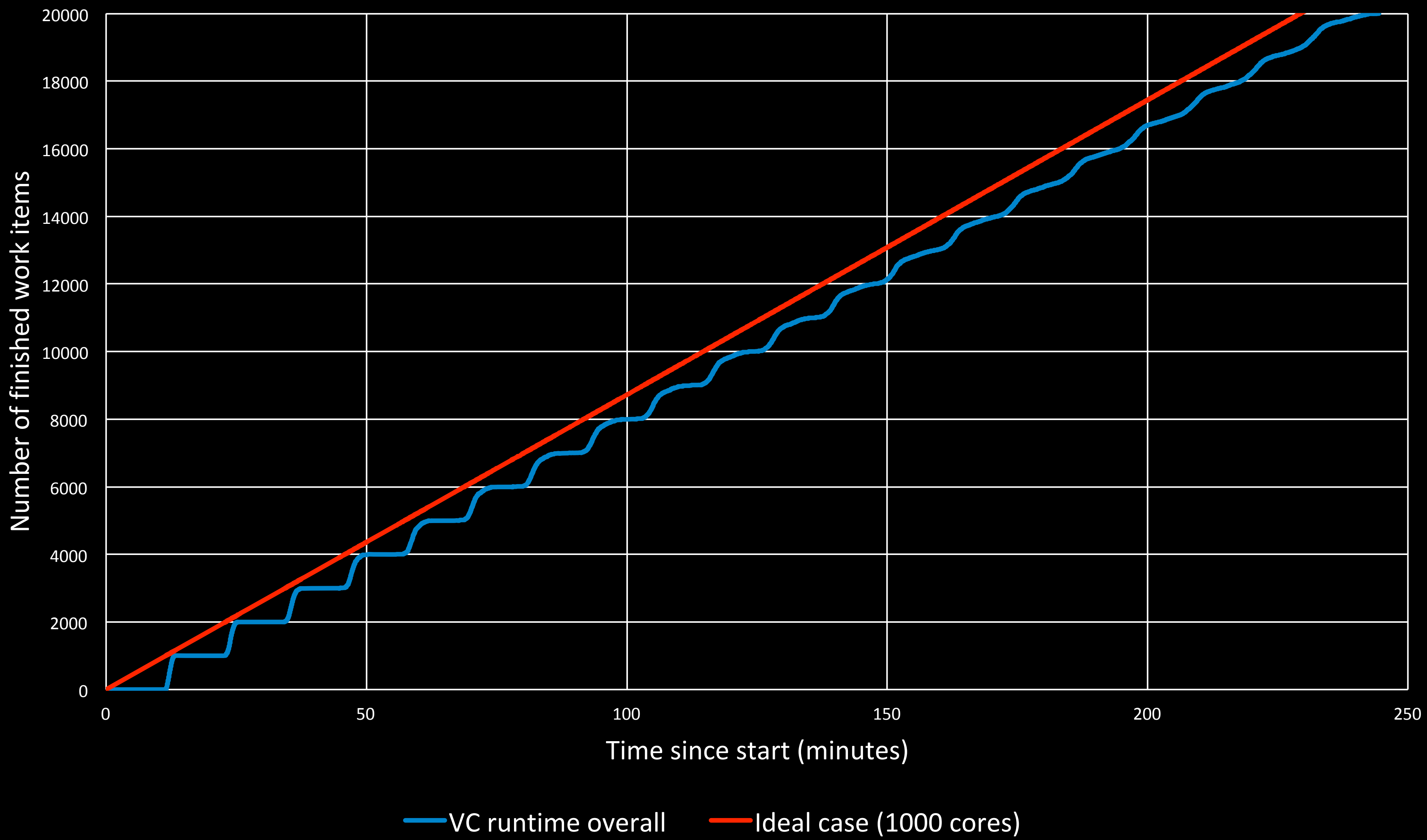
Time line

L32D Throughput with Failure

3: Can Multiple VCPools scale simultaneously?

- Start 10 machines in reliable layer. Each creates a VCPool and manages 100 VCWorkers.
- 10 VCPools assign work items, receive results simultaneously.

Overall L32D Throughput with 10 VCPools



Conclusion

- Scalability test (Experiment 1): a single VCPool can scale to thousands of VCWorkers with negligible overhead
- Failure test (Experiment 2): VCPool can make progress and scale with presence of failure
- Multiple VCPools test (Experiment 3): multiple VCPools can scale simultaneously
- Expectation: hundreds of machines in reliable layer could potentially manage hundreds of thousands of transient machines.

Future Work

- Make AIM project scale to hundreds of thousands of transient machines
- Programming API for high-level operations
  - E.g., automatically support relational operations (Select) in DryadLINQ
- Introduce virtual core into Azure.

Thanks