Making verifiable computation a systems problem

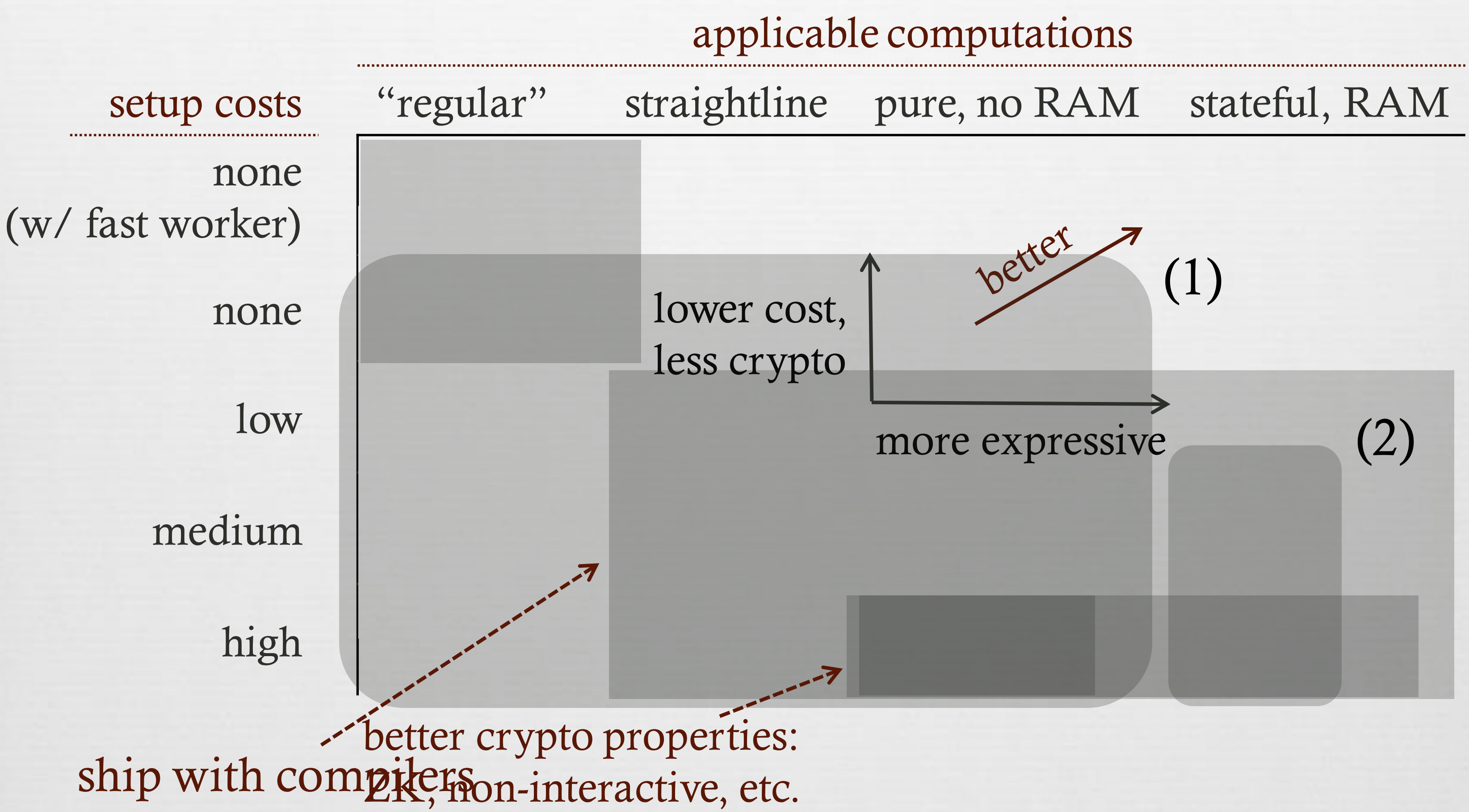
Michael Walfish

The University of Texas at Austin

From a systems perspective, it is an exciting time for this area!

- When we started …
  - … there were no implementations
  - … my colleagues thought I was a lunatic
- Today …
  - … there is a rich design space
  - … the work can be called “systems” with a straight face

A key trade-off is performance versus expressiveness.



(Includes only implemented systems.)

We investigate:

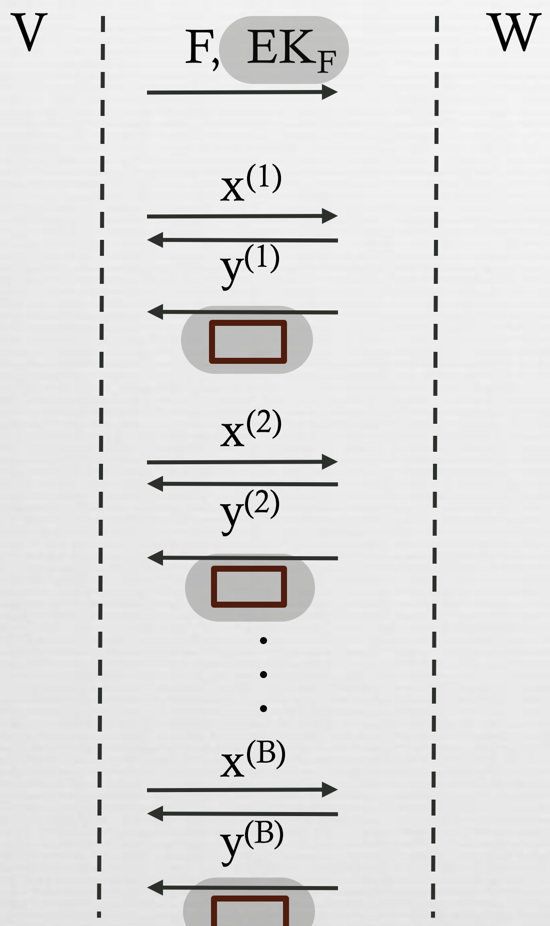
- What are the verifiers' variable (verification, per-instance) costs, and how do they compare to native execution?
- What are the verifiers' fixed (per-computation or per-batch setup) costs, and how do they amortize?
- What are the workers' overheads?

Experimental setup and ground rules

- A system is included iff it has published experimental results.
- Data are from our re-implementations and match or exceed published results.
- All experiments are run on the same machines (2.7Ghz, 32GB RAM). Average 3 runs (experimental variation is minor).
  - For a few systems, we extrapolate from detailed microbenchmarks
- Measured systems:
  - General-purpose: IKO, Pepper, Ginger, Zaatar, Pinocchio
  - Special-purpose: CMT, Pepper-tailored, Ginger-tailored, Allspice
- Benchmarks: 150×150 matrix multiplication and clustering algorithm (others in our papers)

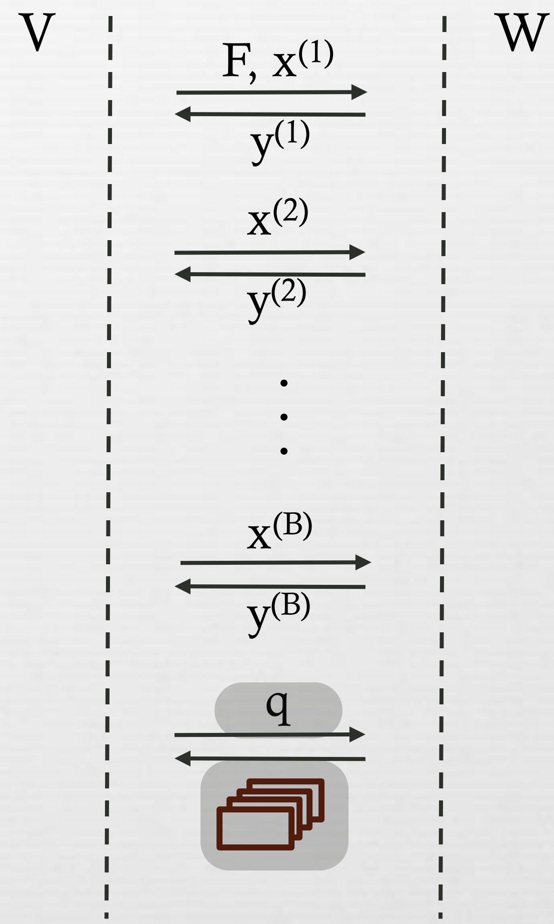
Pinocchio

setup costs are per-computation



Pepper, Ginger, Zaatar

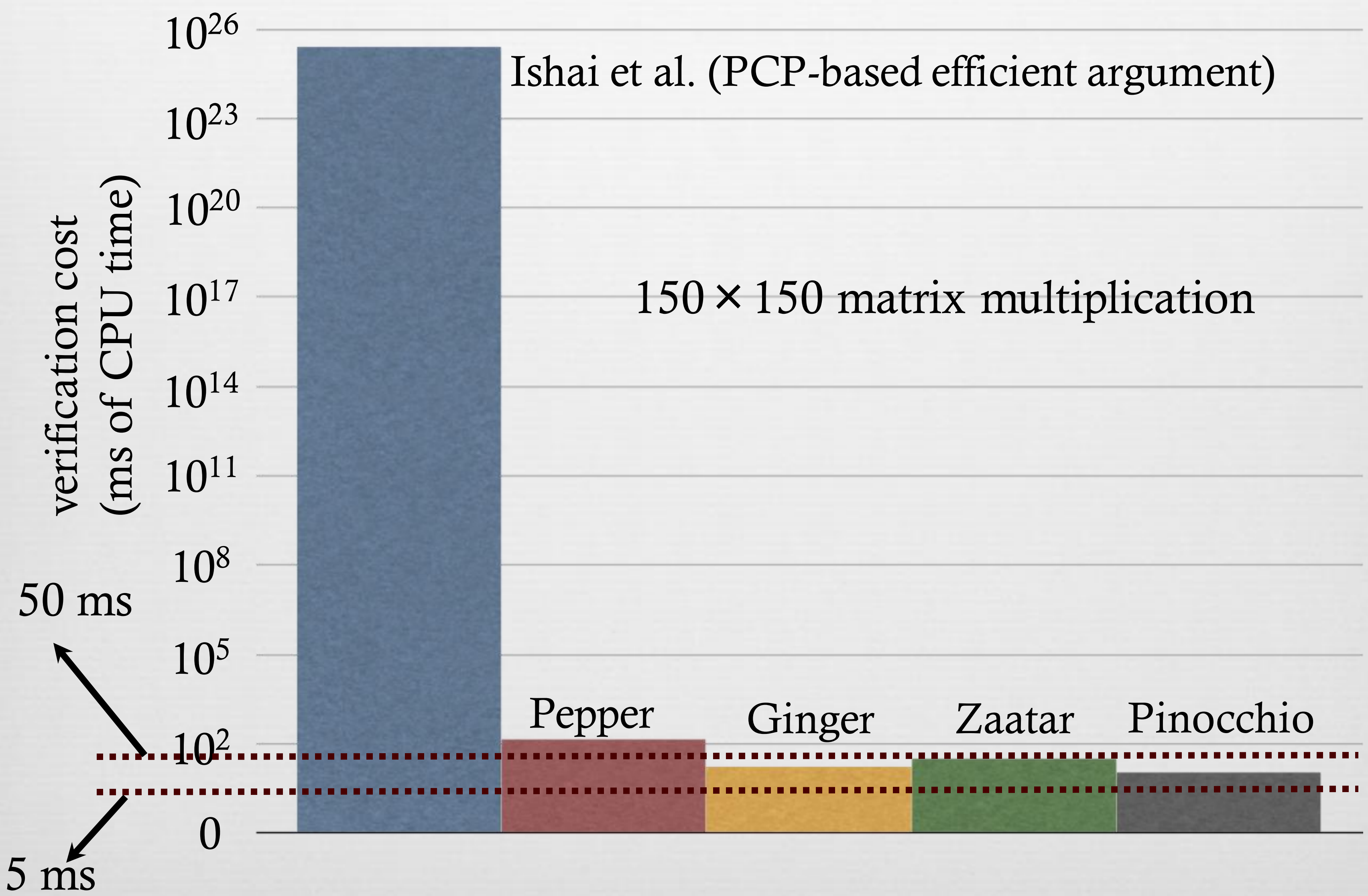
setup costs are per-batch



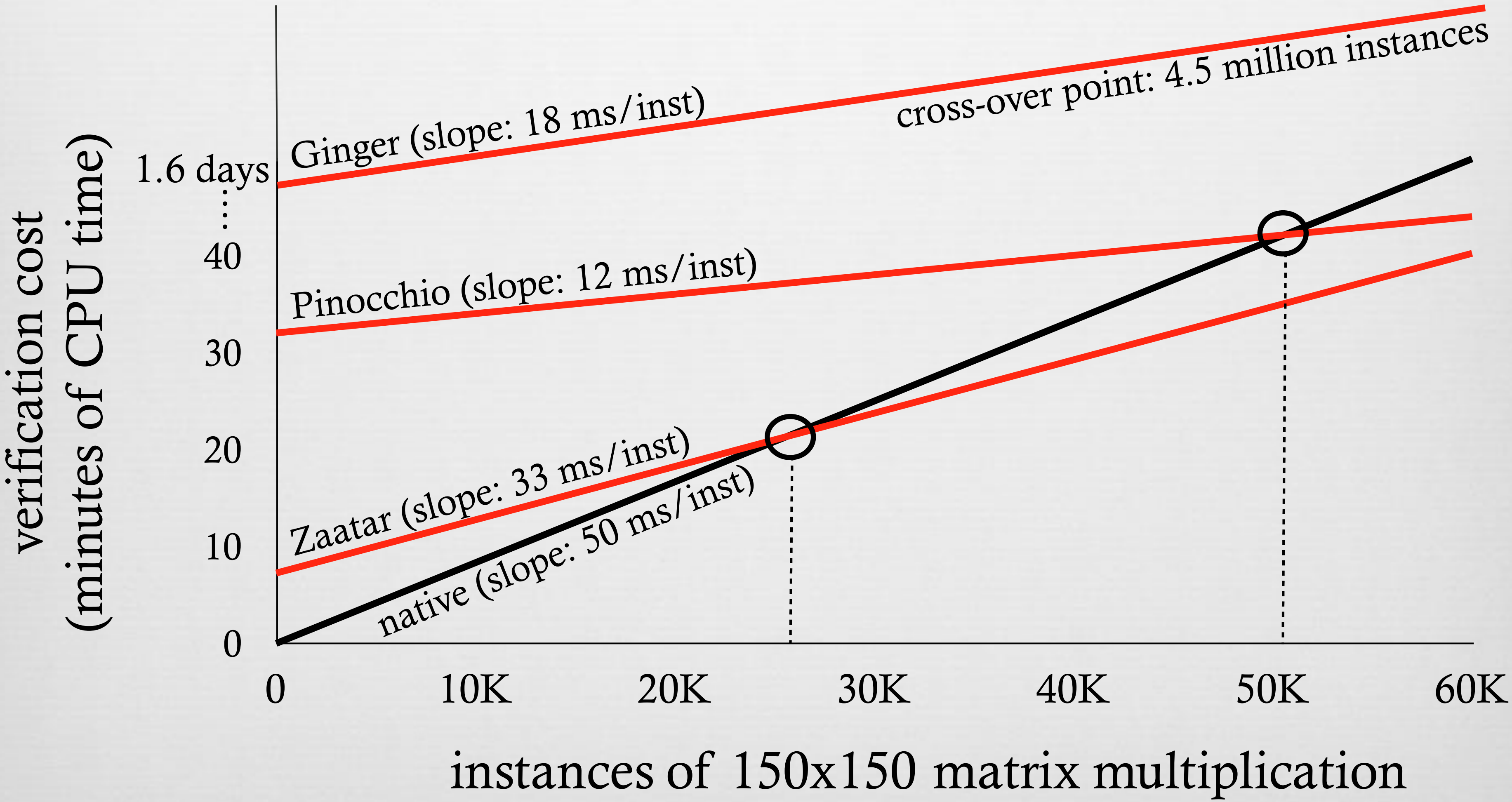
Experimental setup and ground rules

- A system is included iff it has published experimental results.
- Data are from our re-implementations and match or exceed published results.
- All experiments are run on the same machines (2.7Ghz, 32GB RAM). Average 3 runs (experimental variation is minor).
  - For a few systems, we extrapolate from detailed microbenchmarks
- Measured systems:
  - General-purpose: IKO, Pepper, Ginger, Zaatar, Pinocchio
  - Special-purpose: CMT, Pepper-tailored, Ginger-tailored, Allspice
- Benchmarks: 150×150 matrix multiplication and clustering algorithm (others in our papers)

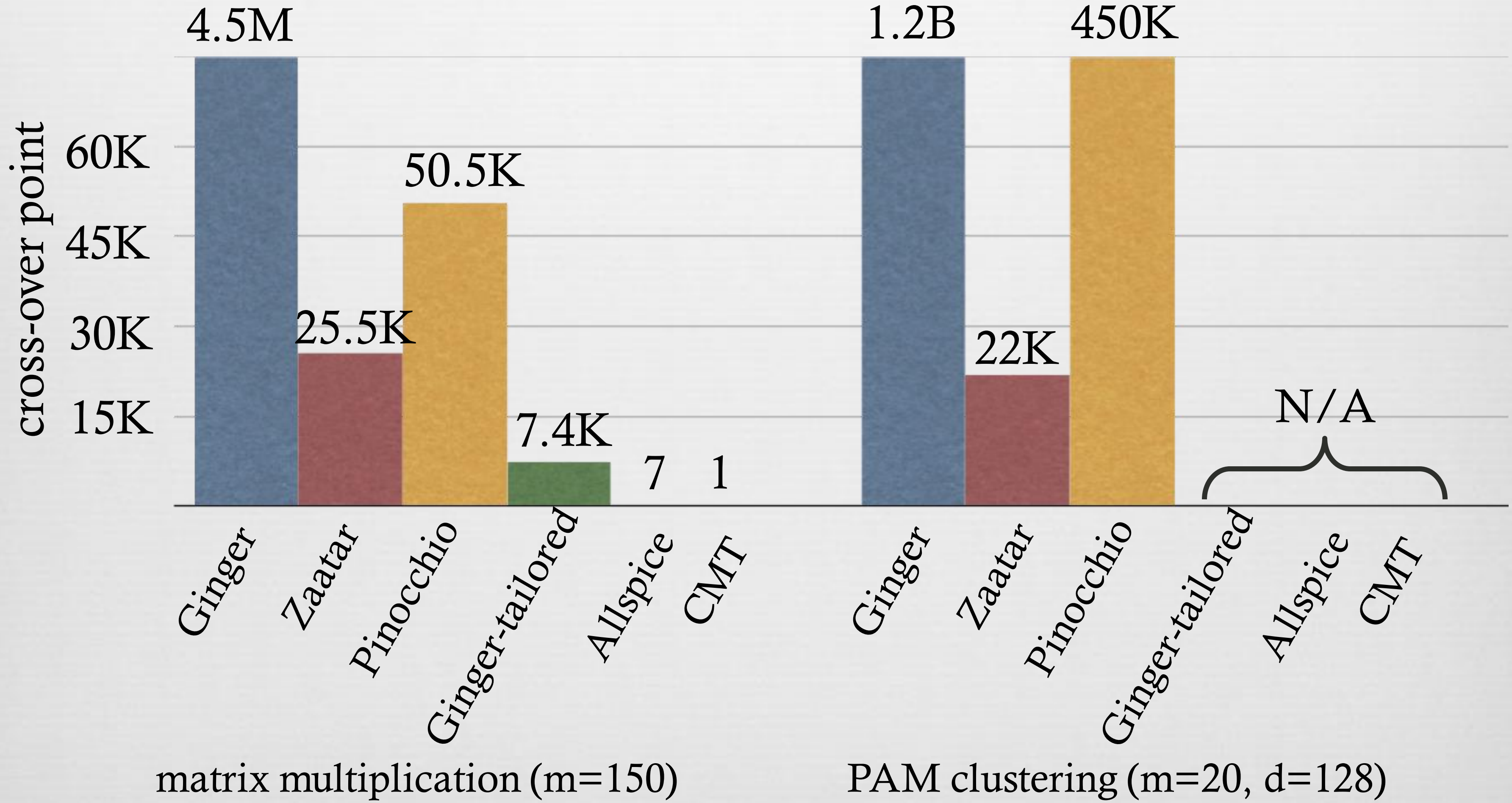
Verification cost sometimes beats (unoptimized) native execution.



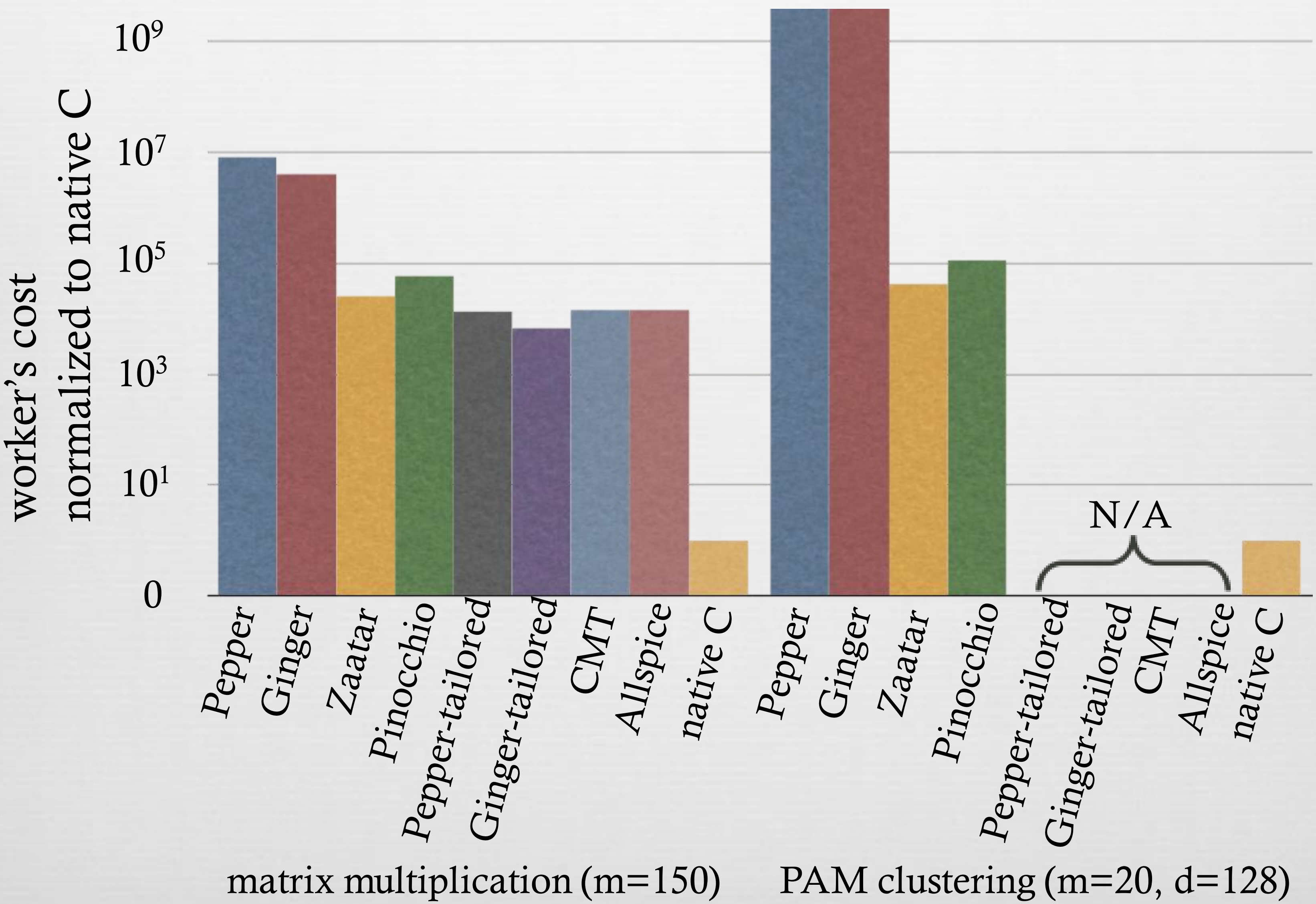
Some of the general-purpose protocols have reasonable cross-over points.



The cross-over points can sometimes improve with special-purpose protocols.



The worker's costs are pretty much preposterous.



Summary of performance in this area

- None of the systems is at true practicality

- Worker's costs still a disaster (though lots of progress)

- Verifier gets close to practicality, with special-purposeness
  - Otherwise, there are setup costs that must be amortized
  - (We focused on CPU; network costs are similar.)

setup costs	applicable computations: "regular"	straightline	pure, no RAM	stateful, RAM
none (w/ fast worker)	Thaler [CRYPTO13]			
none	CMT [ITCS12]			
low		Allspice [Oakland13]		
medium	Pepper [NDSS12]	Ginger [Security12]	Zaatar [Eurosys13]	Pantry [SOSP13]
high			Pinocchio [Oakland13]	Pantry [SOSP13]

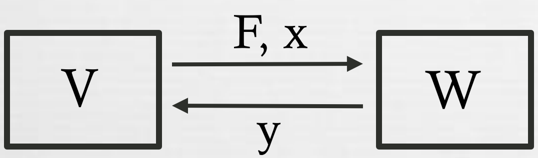
better

(1)

(2)

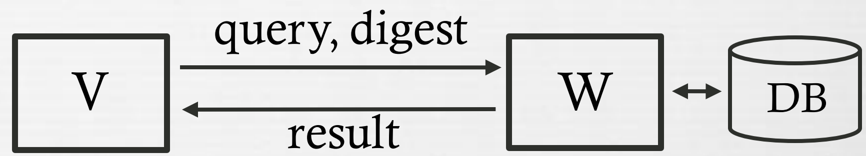
Pantry [SOSP13] creates verifiability for real-world computations

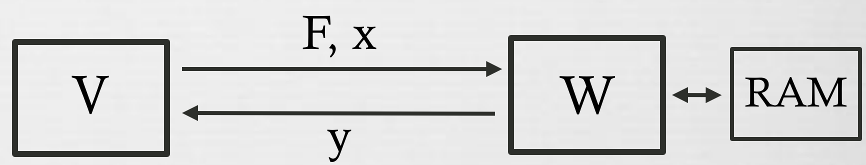
before:

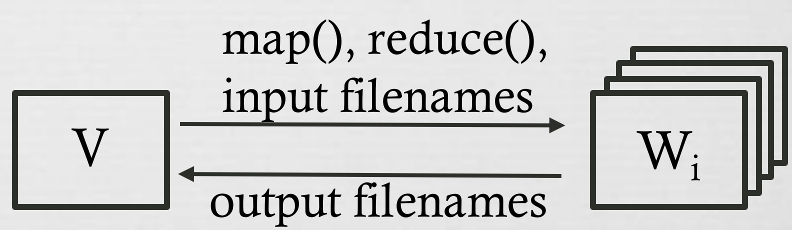


- V supplies all inputs
- F is pure (no side effects)
- All outputs are shipped back

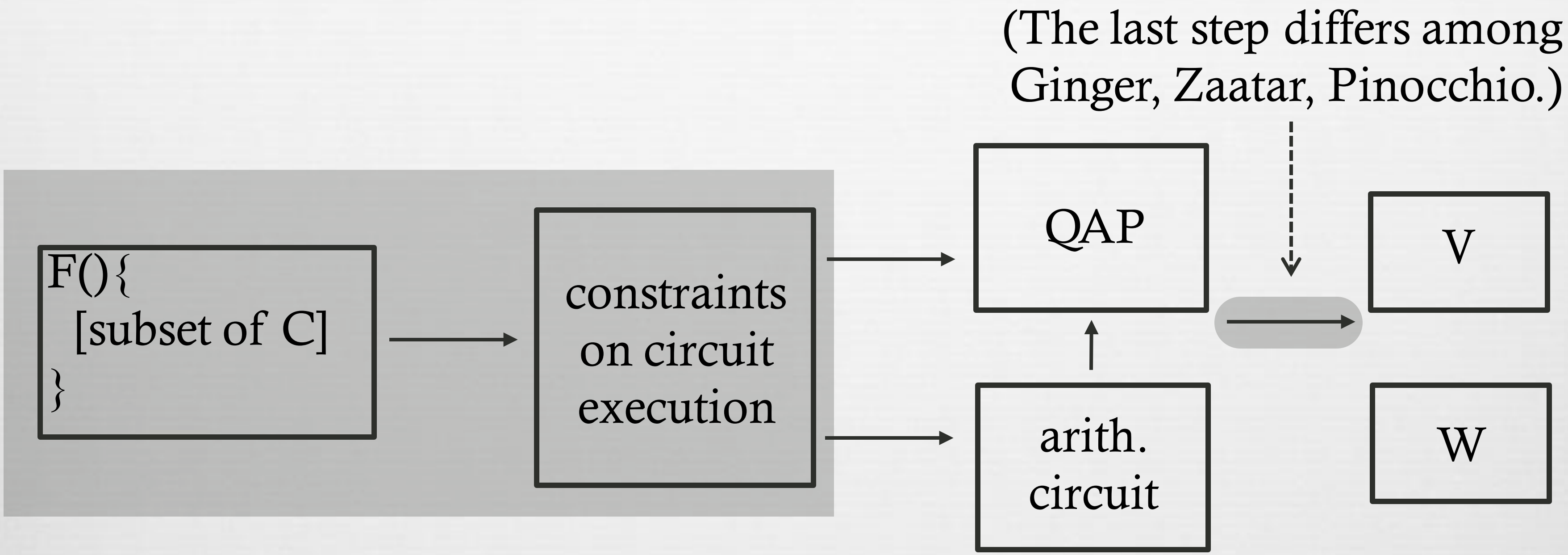
after:

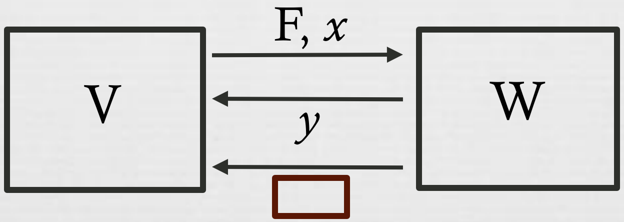






Recall the compiler pipeline.





Programs compile to constraints on circuit execution.

dec-by-three.c

```
f(X) {
  Y = X – 3;
  return Y;
}
```

compiler →

$$\left\{ \begin{array}{l} 0 = Z - X, \\ 0 = Z - 3 - Y \end{array} \right\}$$

Input/output pair correct ⇔ constraints satisfiable.

As an example, suppose X = 7.

if Y = 4 …

$$\left\{ \begin{array}{l} 0 = Z - 7 \\ 0 = Z - 3 - 4 \end{array} \right\}$$

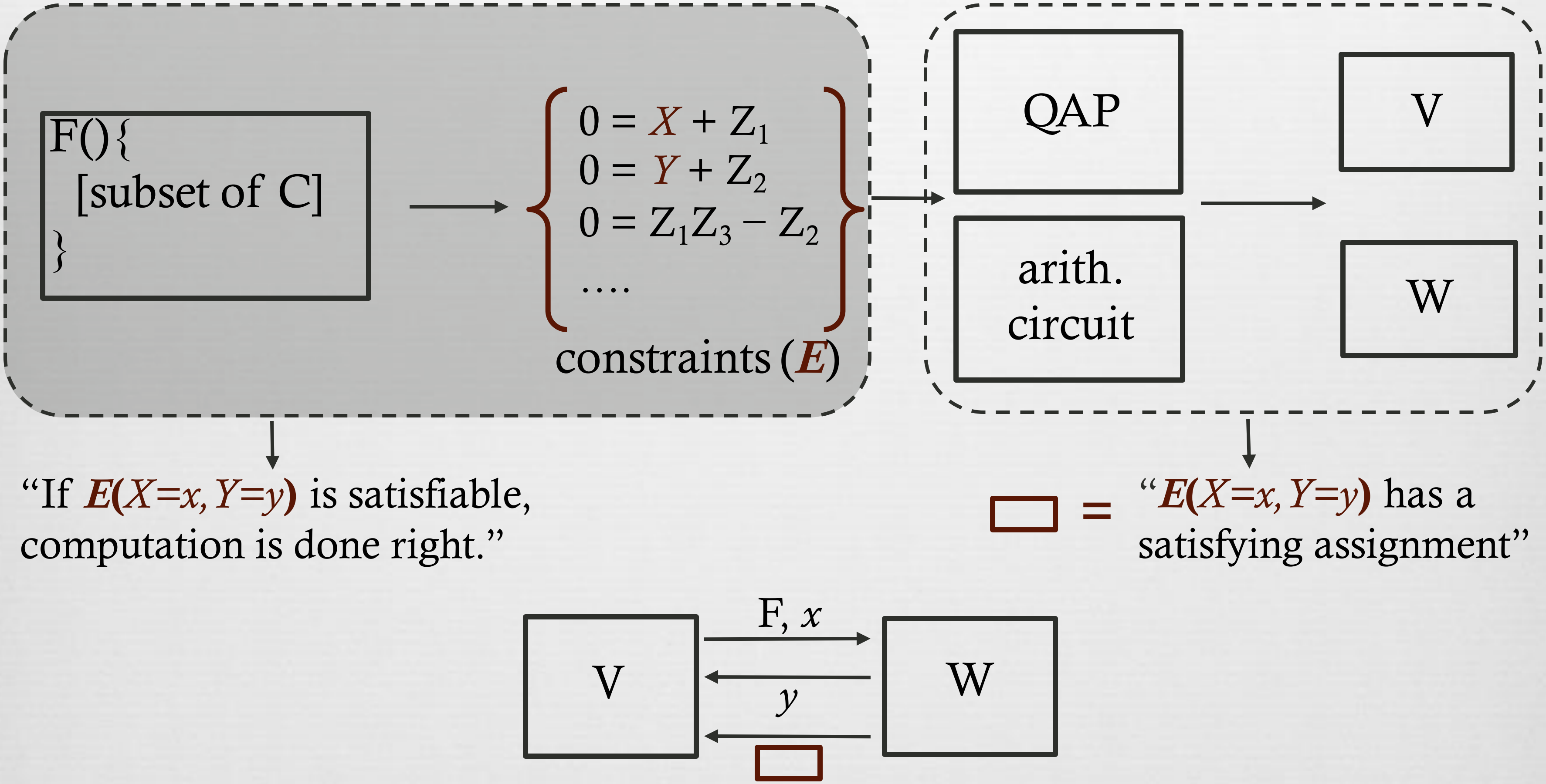
… there is a solution

if Y = 5 …

$$\left\{ \begin{array}{l} 0 = Z - 7 \\ 0 = Z - 3 - 5 \end{array} \right\}$$

… there is no solution

The pipeline decomposes into two phases.



Design question: what can we put in the constraints so that satisfiability implies correct storage interaction?

How can we represent storage operations? (1)

Representing “load(addr)” explicitly would be horrifically expensive.

Straw man: variables M_0, …, M_{size} contain state of memory.

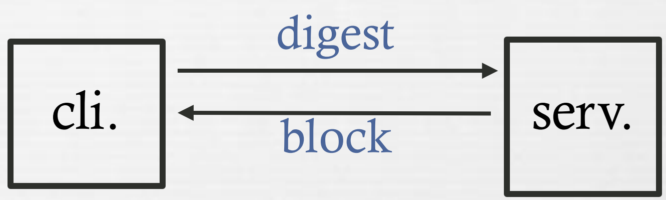
$$B = load(A) \longrightarrow \left\{ \begin{array}{l} B = M_0 + (A - 0) \bullet F_0 \\ B = M_1 + (A - 1) \bullet F_1 \\ B = M_2 + (A - 2) \bullet F_2 \\ \ldots \\ B = M_{size} + (A - size) \bullet F_{size} \end{array} \right\}$$

Requires two variables for every possible memory address!

How can we represent storage operations? (2)

Consider self-certifying blocks:

$$\text{hash(block)} \stackrel{?}{=} \text{digest}$$



- They bind references to values
- They provide a substrate for verifiable RAM, file systems, …
  [Merkle CRYPTO87, Fu et al. OSDI00, Mazières & Shasha PODC02, Li et al. OSDI04]

Key idea: encode the hash checks in constraints

- This can be done (reasonably) efficiently

Folklore: “this should be doable.” (Pantry’s contribution: “it is.”)

We augment the subset of C with the semantics of untrusted storage

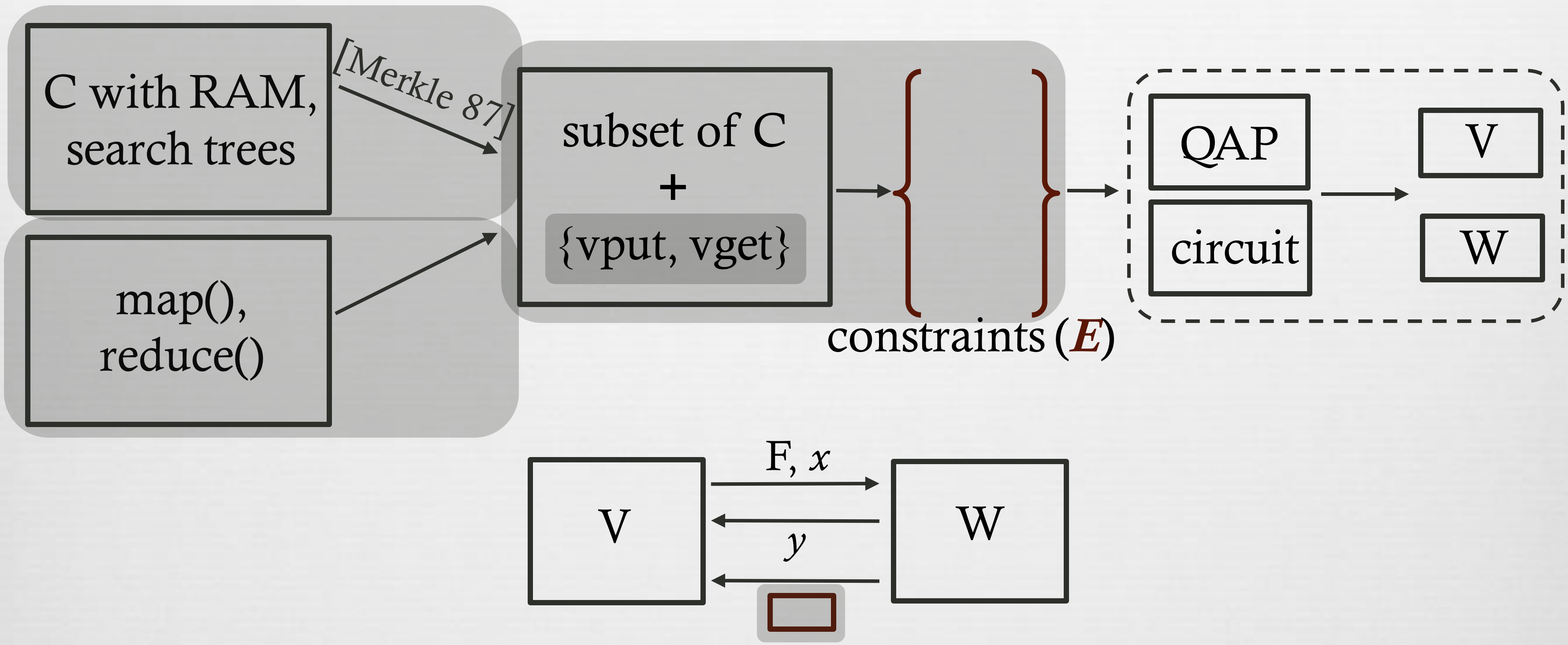
- block = vget(digest): retrieves block that must hash to digest
- hash(block) = vput(block): stores block; names it with its hash

```
add_indirect(digest d, value x)
{
   value z = vget(d);
   y = z + x;
   return y;
}
```

$$\longrightarrow \left\{ \begin{array}{c} \left\{ d = \text{hash}(Z) \right\} \\ y = Z + x \end{array} \right\}$$

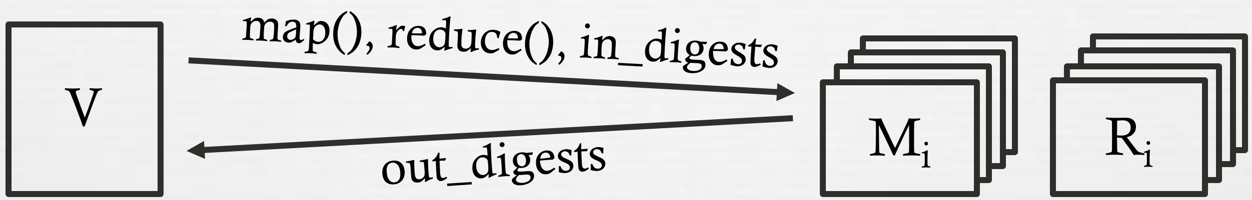
Worker is obliged to supply the "correct" Z (meaning something that hashes to d).

Putting the pieces together

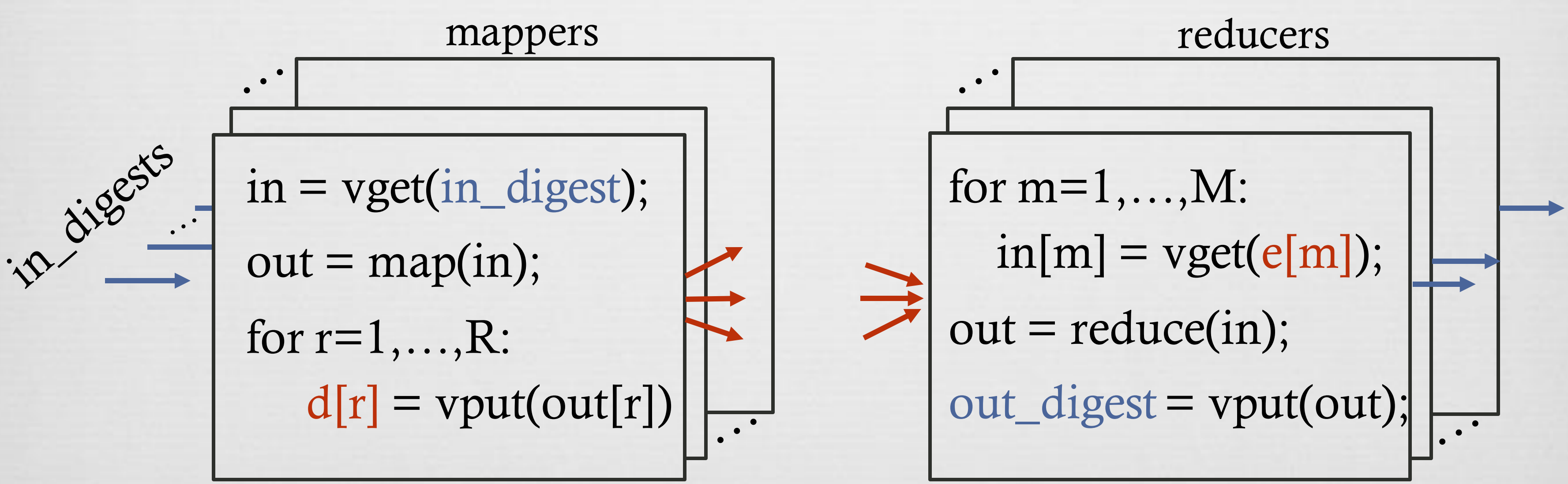


- recall: ▭ = “I know a satisfying assignment to ***E***(X=x, Y=y)”
- checks-of-hashes pass ⇔ satisfying assignment identified
- checks-of-hashes pass ⇔ storage interaction is correct
- storage abstractions can be built from {vput(), vget()}

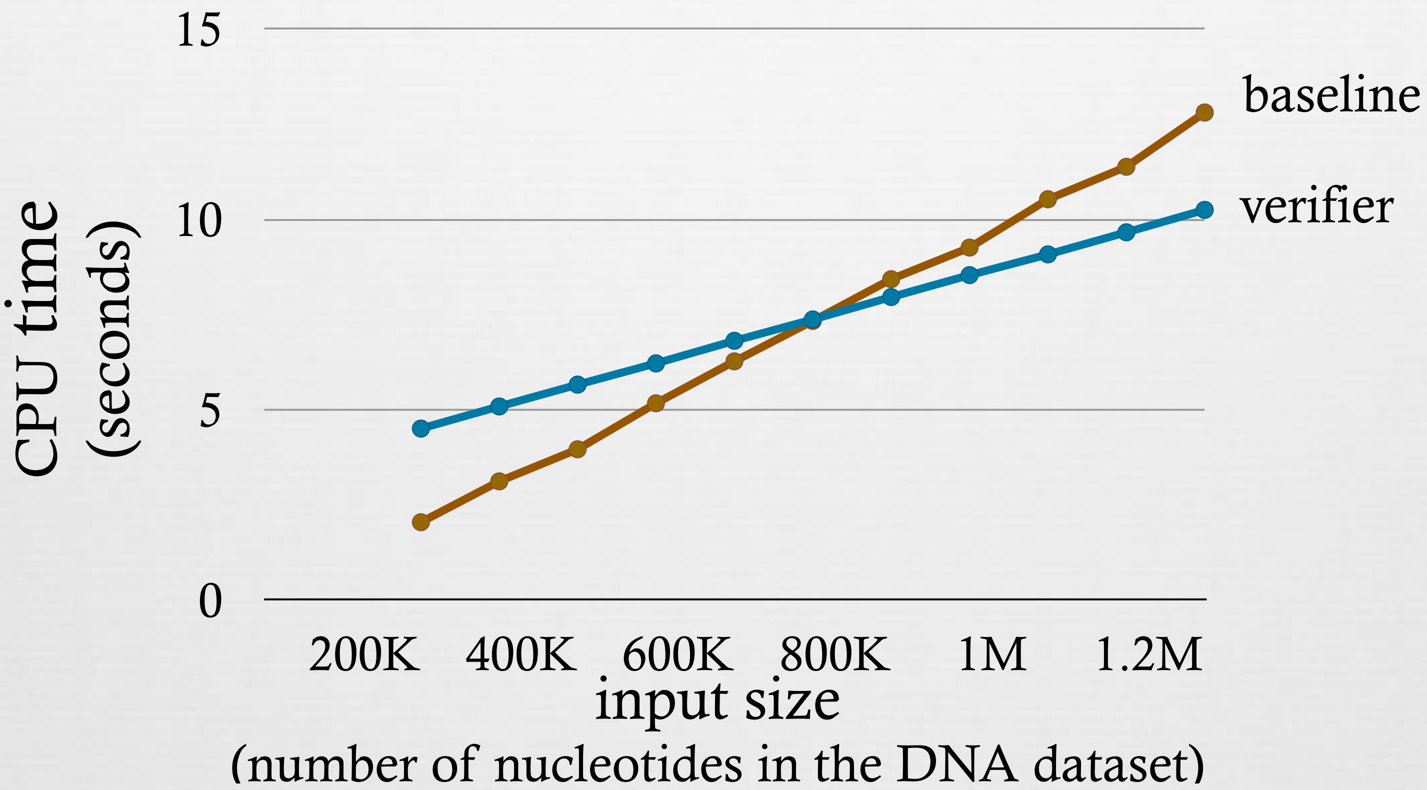
The verifier is assured that a MapReduce job was performed correctly—without ever touching the data.



The two phases are handled separately:



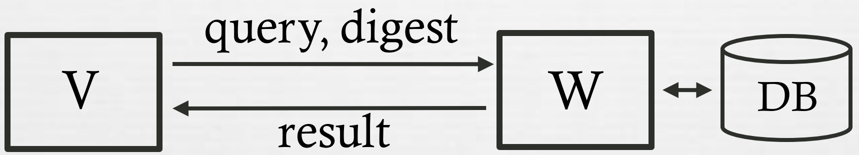
Example: for a DNA subsequence search, the verifier saves work, relative to performing the computation locally.



- A mapper gets 1000 nucleotides and outputs matching locations
- Vary mappers from 200 to 1200; reducers from 20 to 120

Pantry applies fairly widely

- Our implemented applications include:



  - Verifiable queries in (highly restricted) subset of SQL
  - Privacy-preserving facial recognition

- Our implementation works with Zaatar and Pinocchio

Major problems remain for this area

- Setup costs are high (for the general-purpose systems)
- Verification costs are high, relative to native execution
  - Evaluation baselines are highly optimistic
  - Example:100×100 matrix multiplication takes 2 ms on modern hardware; no VC system beats this.
- Worker overhead is 1000×
- The computational model is a toy
  - Loops are unrolled, memory operations are expensive

Summary and take-aways

- A framework for organizing the research in this area is performance versus expressiveness.
- Pantry extends verifiability to stateful computations, including MapReduce, DB queries, RAM, etc.
- Major problems remain for all of the systems
  - Setup costs are high (for the general-purpose systems), and verification does not beat optimized native execution
  - Worker costs are too high, by many orders of magnitude
  - The computational model is a toy