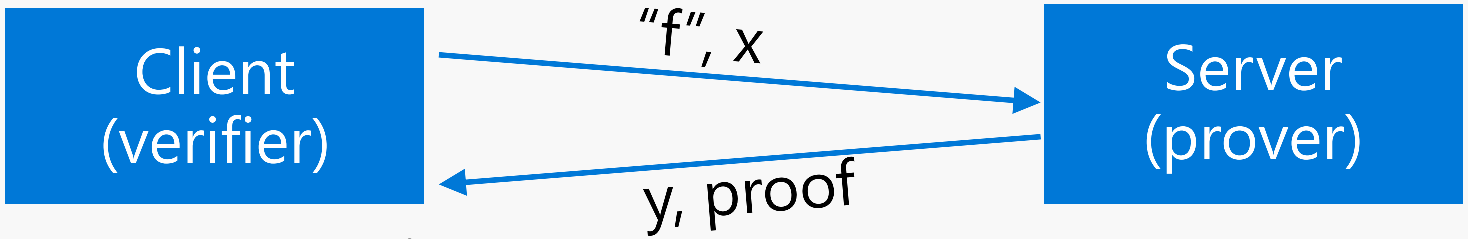
Implementations of probabilistic proofs for verifiable outsourcing: survey and next steps

Srinath Setty
Microsoft Research

(Thanks to Michael Walfish for some of the slides.)



without executing f,
can check that: "y = f(x)"
more generally: "prover knows w s.t. y = f(x,w)"

Motivation: Third-party computing

- Cloud computing, distributed ledger technologies (DLTs)

Requirements:

- Efficiency (client CPU, communication, server CPU, etc.)
- Privacy of w (zero-knowledge, desirable in some applications)

GMR85
BCC88
Kilian92
Micali94
BG02
GOS06
IKO07
GKR08
CKV10
GGP10
Groth10
Lipmaa12
GGPR12
BCCT13

A naïve implementation of the theory results in outrageous costs

Thousands of CPU years to verifiably execute even simple computations

What do we need?

Practicality (as real people understand the term) in addition to efficiency and privacy for w

Good news

- Running code; cost reductions of 10^{20} vs. theory
- Compilers from C to protocol entities
- Stateful computations; remote inputs, outputs
- Concretely efficient verifiers

Bad news

- Extreme expense: 10^6x overhead vs. native
- Programming model is clumsy
- Useful only for special-purpose applications

SBW11
CMT12
SMBW12
TRMP12
SVPBBW12
SBVBPW13
VSBW13
PGHR13
Thaler13
BCGTV13
BFRSBW13
BFR13
DFKP13
BCTV14a
BCTV14b
BCGGMTV14
FL14
KPPSST14
FGV14
BBFR14
WSRHBW15
CFHKKNPZ15
CTV15
WHGSW16
DFKP16
FFGKOP16
ZGKPP17
WJBSTWW17

Note: There are pragmatic alternatives

Replication [Castro & Liskov TOCS02]

Far less expensive, but it does not support privacy for w

Trusted hardware such as Intel SGX

Far less expensive, but requires significant trust

No formal security guarantees

Hard (or impossible) to reason about end-to-end security

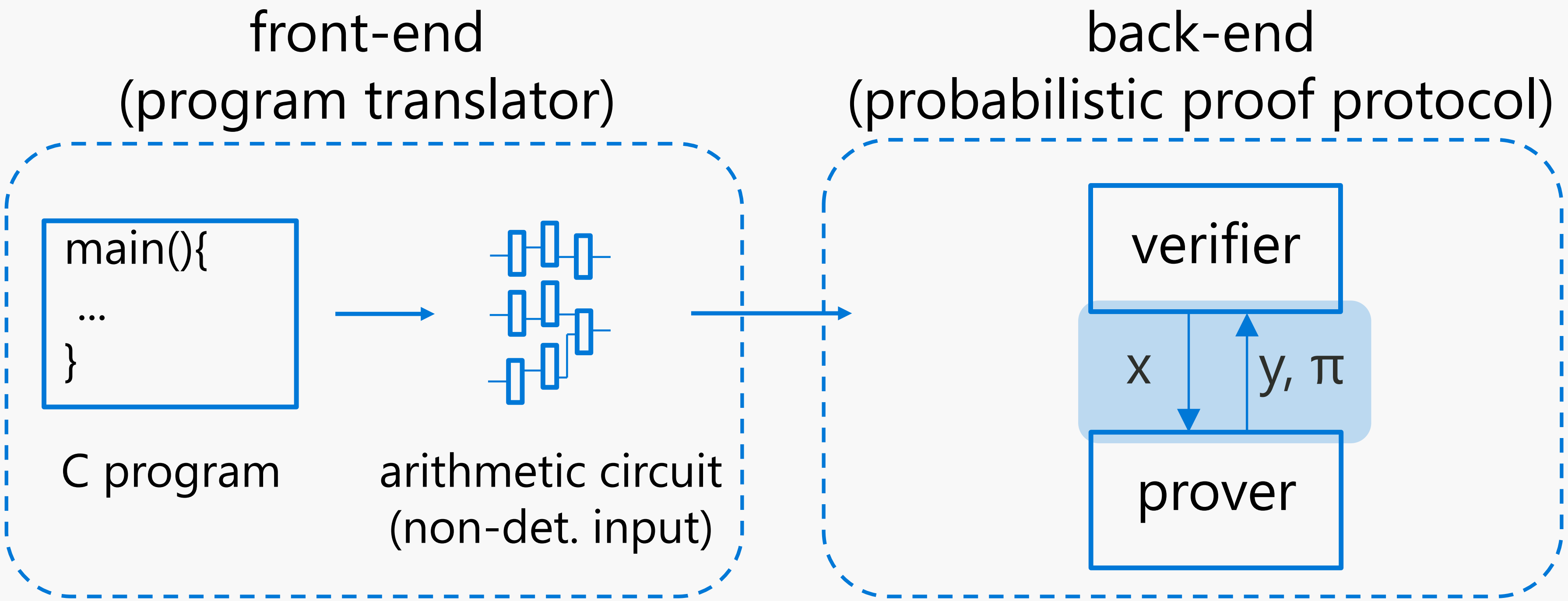
Rest of this talk

Summary of state of the art implementations

Reality check with a performance evaluation

Next steps

Common framework in state of the art systems



General "processor"

Custom circuit

interactive proof [GKR08]

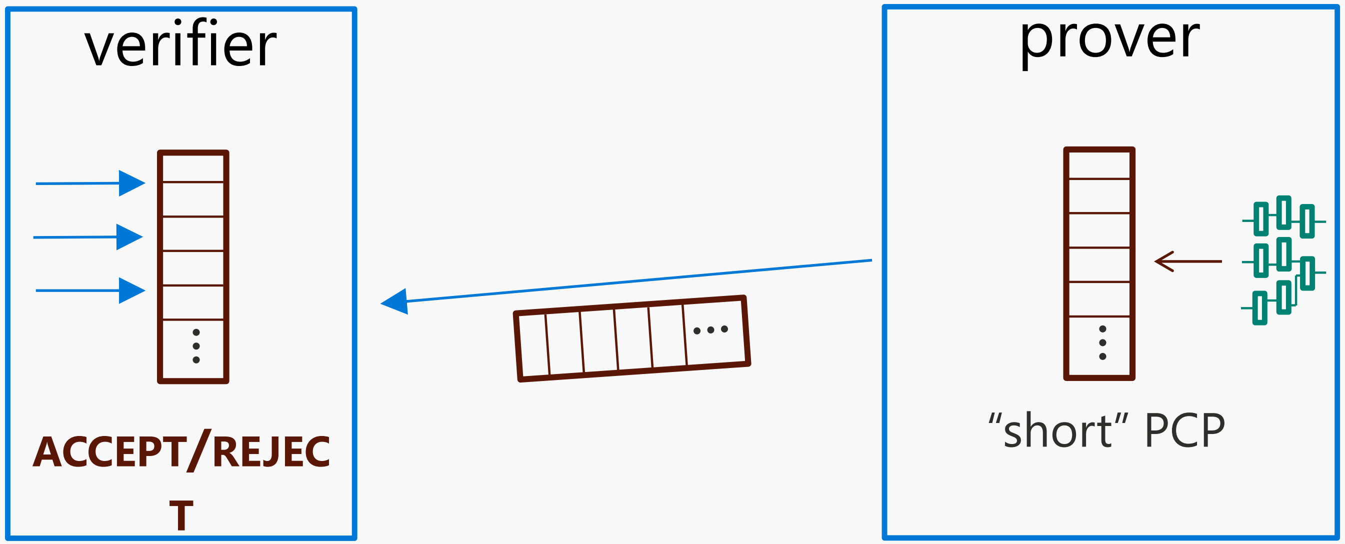
interactive argument [IKO07]

non-interactive argument [Groth10, Lipmaa12, GGPR12]

	Interactive proofs [GKR08, CMT12, …]	Arguments: Interactive [IKO07, SBW11, SMBW12, …]	Arguments: Non-interactive [Groth10, Lipmaa12, GGPR12, …]
Circuit type	Deterministic	Non-deterministic	Non-deterministic
#Rounds	Lots	Two	One
Assumptions	None	Simple, falsifiable	Non-standard
Prover expense	10 to 100x	10^6x	10^6x
Verifier setup	0 or (10 to 100x)	10^6x	10^6x
Zero-knowledge	No	No	Yes
Hardware impl.	Yes	Non-amenable	Non-amenable

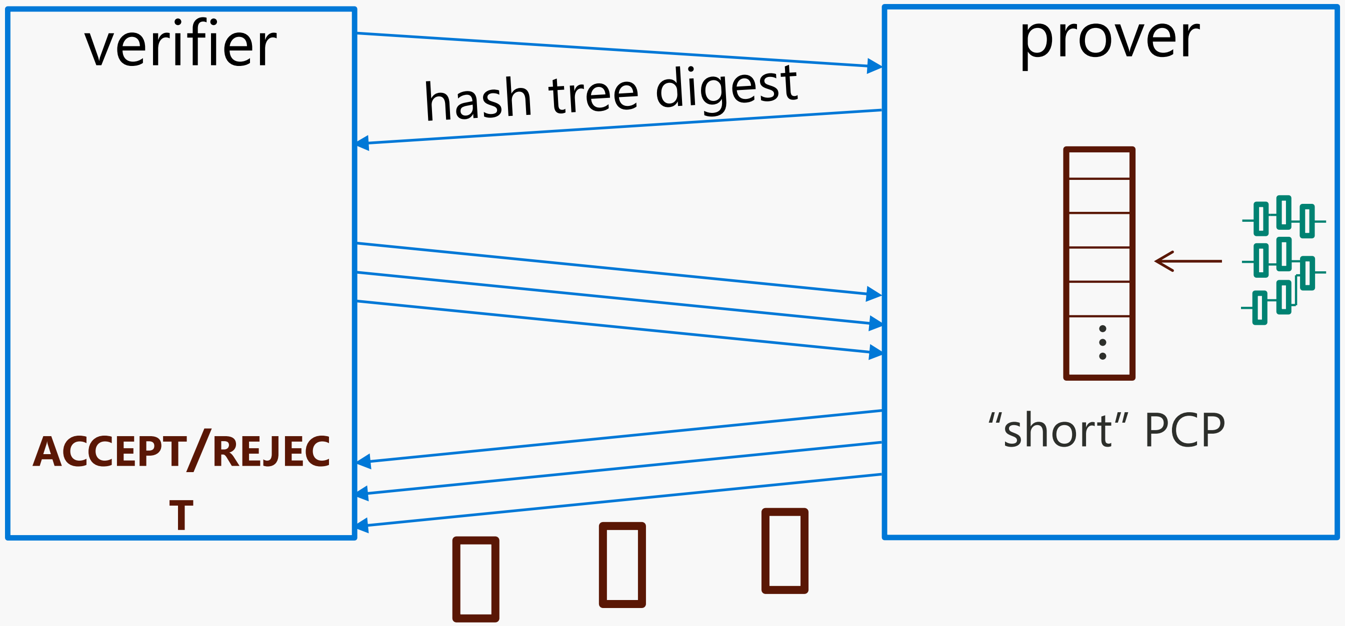
All recent implementations use the QAP encoding [GGPR12]

Attempt 1: Use PCPs that are asymptotically short [ALMSS92, AS92] [BGHSV05, BGHSV06, Dinur07, BS08, Meir12, BCGT13]



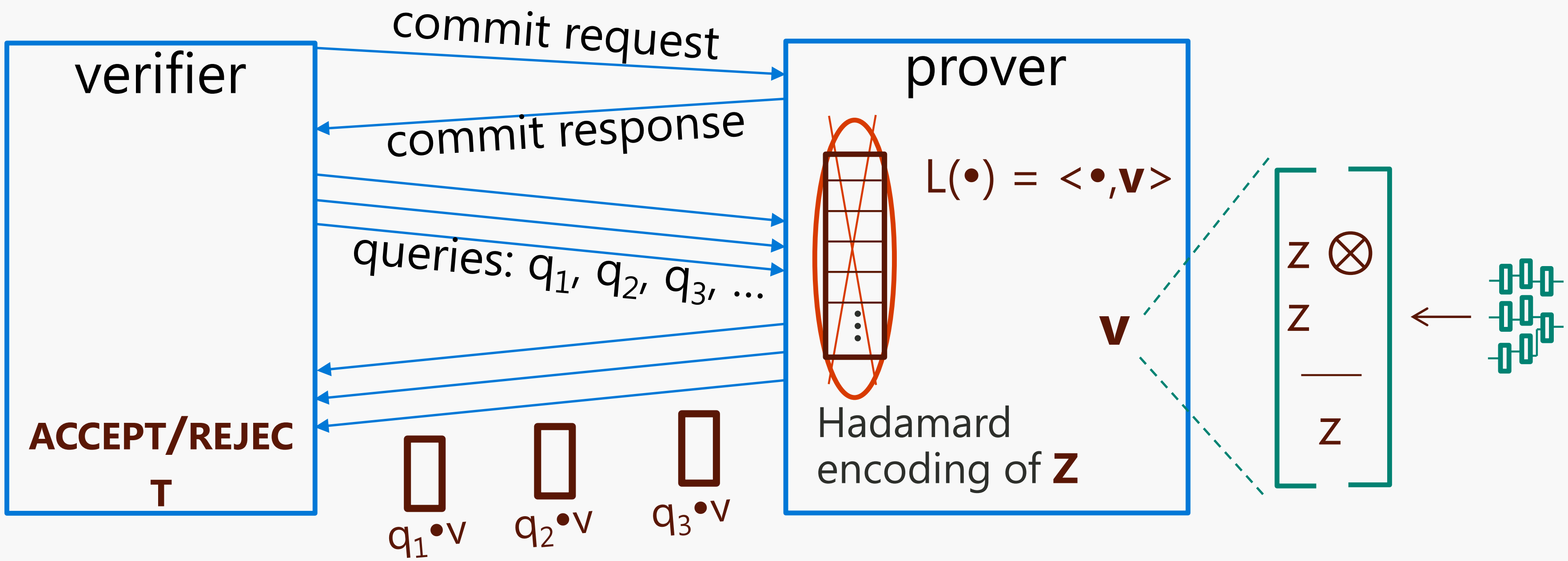
This does not meet the efficiency requirements (because |PCP| > running time of f).

Attempt 2: Use arguments or CS proofs [Kilian92, Micali94]



But the constants seem too high …

Attempt 3: Use long PCPs interactively [IKO07, SMBW12, SVPBBW12]



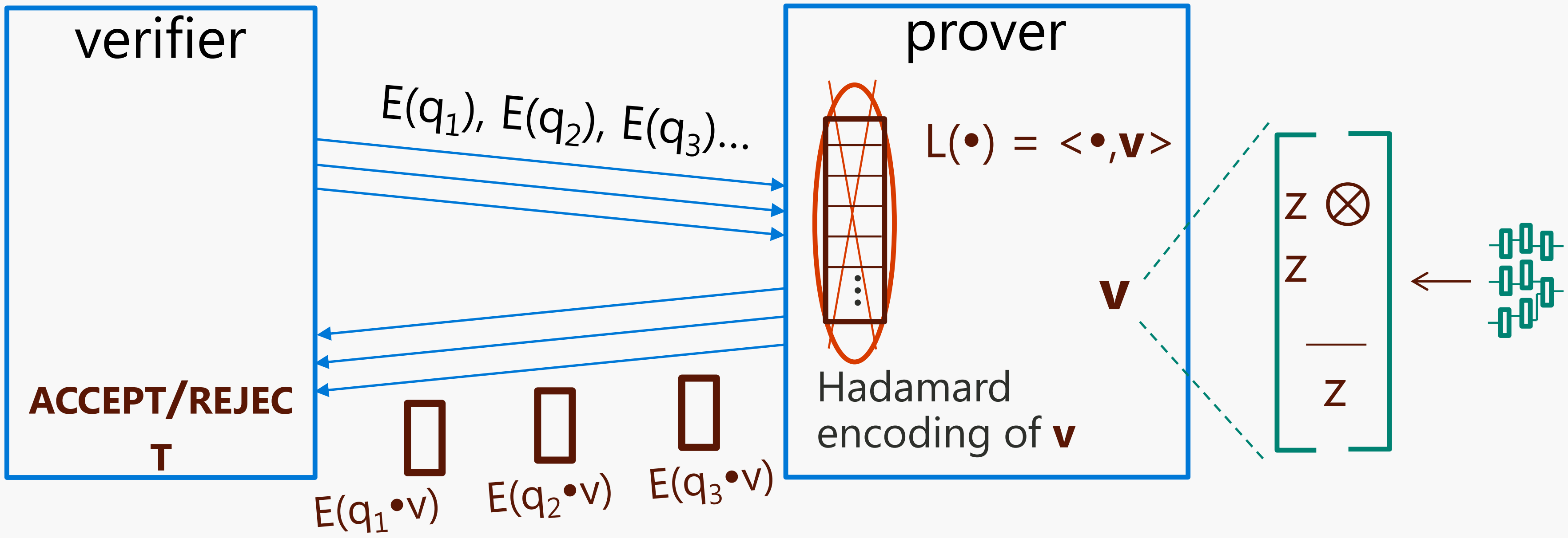
Achieves simplicity, with good constants …

… but pre-processing is required (because $|q_i|=|v|$)

… and prover's work is quadratic; address that shortly

Attempt 4: Use long PCPs non-interactively

[BCIOP13]



Query processing now happens “in the exponent”

… pre-processing still required (again because $|q_i|=|v|$)

… prover’s work still quadratic; addressing that soon

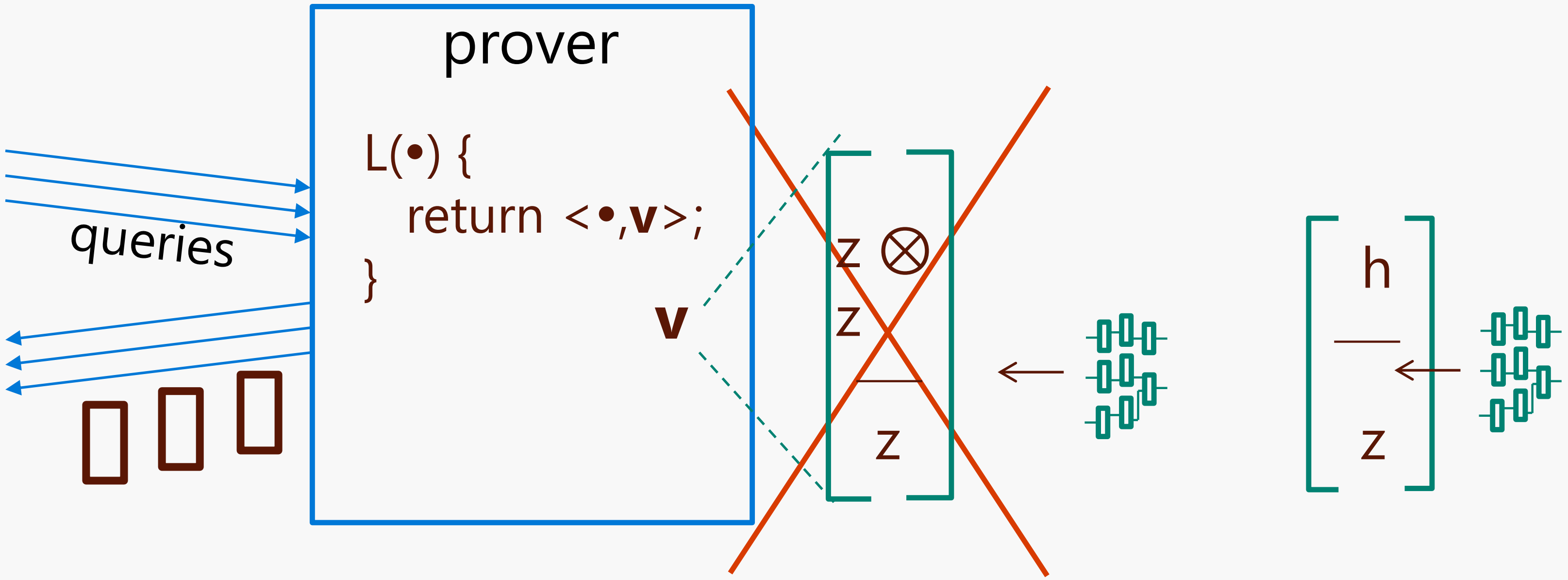
Recap

	efficient (short) PCPs	arguments, CS proofs	arguments w/ preprocessing	SNARGs w/ preprocessing
who	ALMSS92, AS92, BGSHV, Dinur, …	Kilian92, Micali94	IKO07, SMBW12, SVPBBW12, SBVBPW13	GGPR12, BCIOP13, …
what	classical PCP	commit to PCP by hashing	commit to long PCP using linearity	encrypt queries to a long PCP
security	unconditional	CRHFs	linearly HE	knowledge-of-exponent
why/why not	not efficient for V	constants are unfavorable	simple	simple, non-interactive

(Thanks to Rafael Pass.)

Final attempt: apply linear query structure to GGPR's QAPs

[Groth10, Lipmaa12, GGPR12]

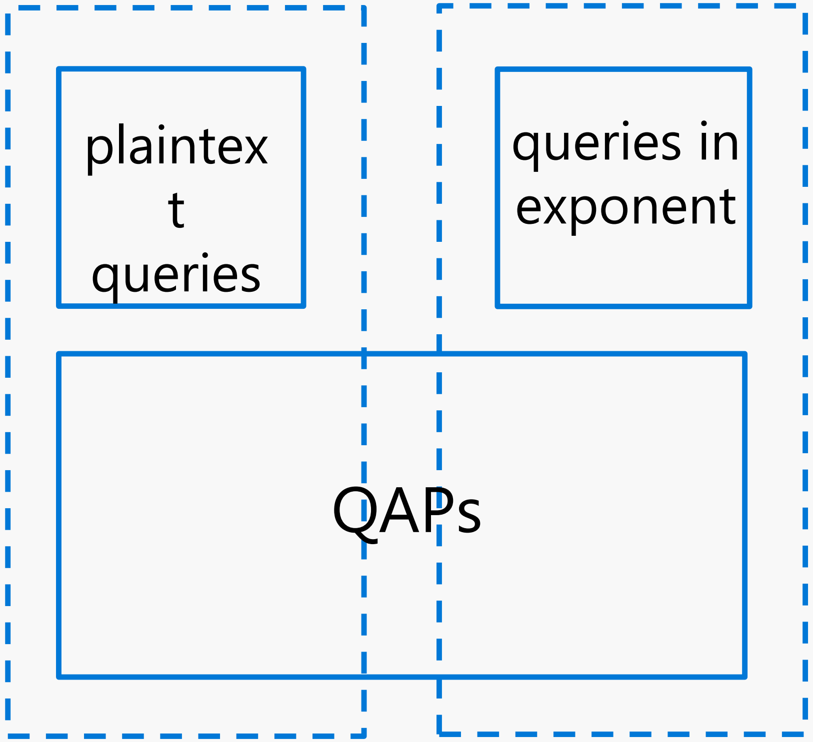


Addresses the issue of quadratic costs.

PCP structure implicit in GGPR. Made explicit in [BCIOP13, SBVBBW13].

"Zaatar"
[SBVBBW13]

interactive argument
[IKO07]



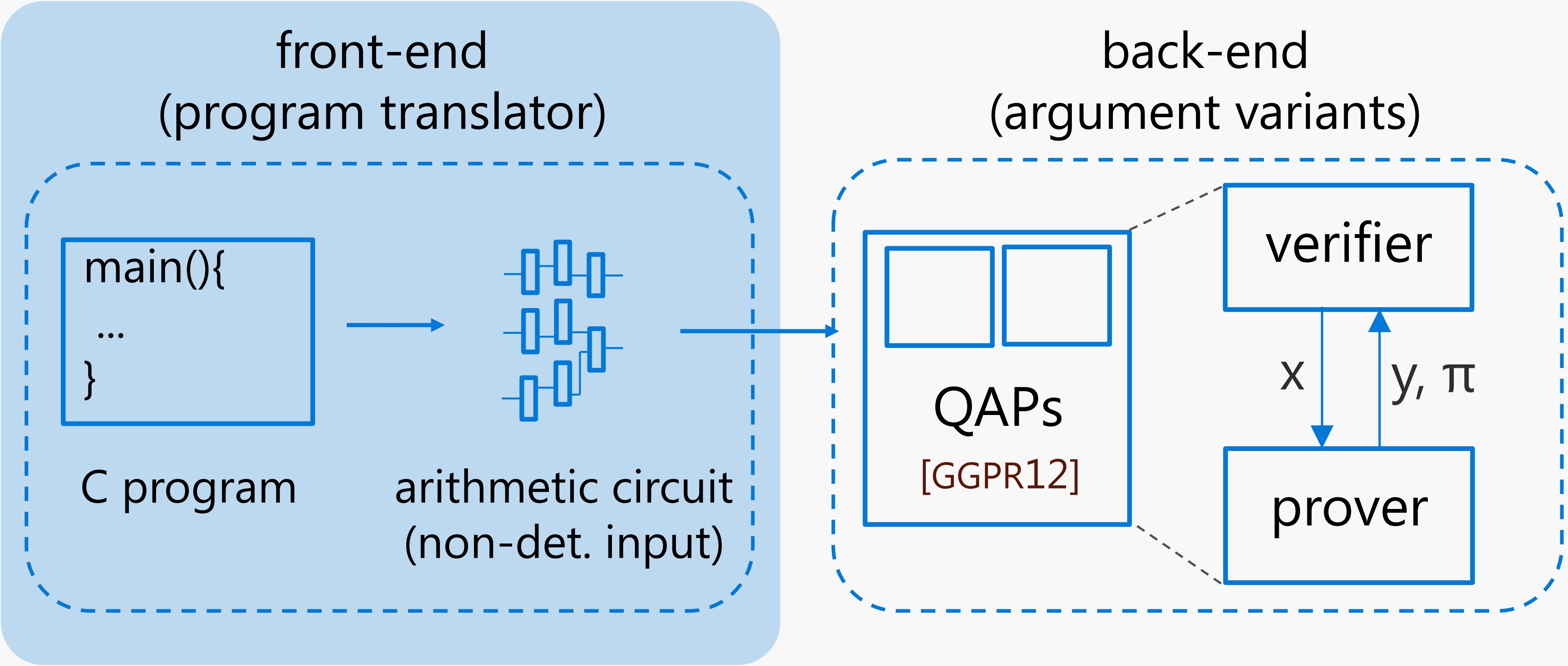
"Pinocchio," "libsnark"
[PGHR13, BCTV14a]

SNARG, zk-SNARK with pre-processing
[Groth10, BCCT12, GGPR12]

pre-processing avoidable in theory
[BCCT13, BCTV14b, CTV15]

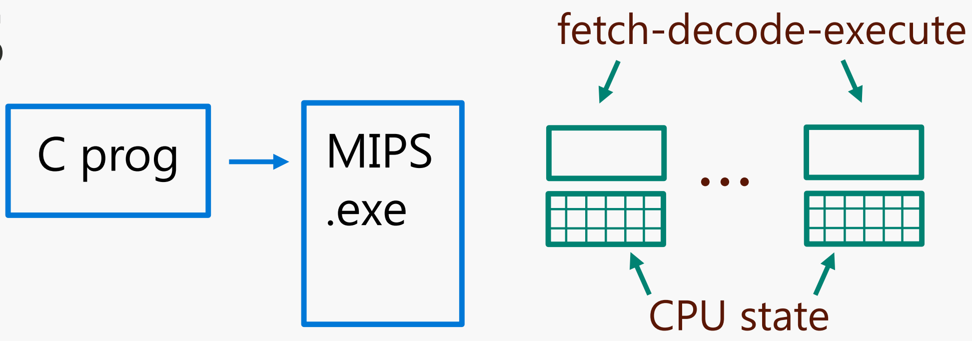
- standard assumptions
- amortize over batch
- interactive

- non-falsifiable assumptions
- amortize indefinitely
- non-interactive, ZK, ...



front-end
(program translator)
main(){
...
}
C program
arithmetic circuit
(non-det. input)
back-end
(argument variants)
QAPs
[GGPR12]
verifier
x
y, π
prover

State of the art front-ends

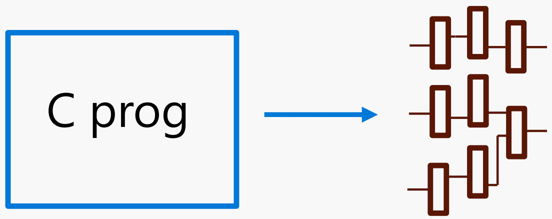


circuit is unrolled CPU execution

[BCGTV13, BCTV14a, BCTV14b, CTV15]

"General" processor [TinyRAM]

- Verbose circuits (costly)
- Good amortization
- Great programmability



each line translates to gates

[SBVBPW13, VSBW13, PGHR13, BFRSBW13, BCGGMTV14, BBFR14, FL14, KPPSST14, WSRBW15, CFHKKNPZ15]

Custom circuits

- Concise circuits
- Amortization worse
- How is programmability?

Front-ends trade off performance and expressiveness

applicable computations

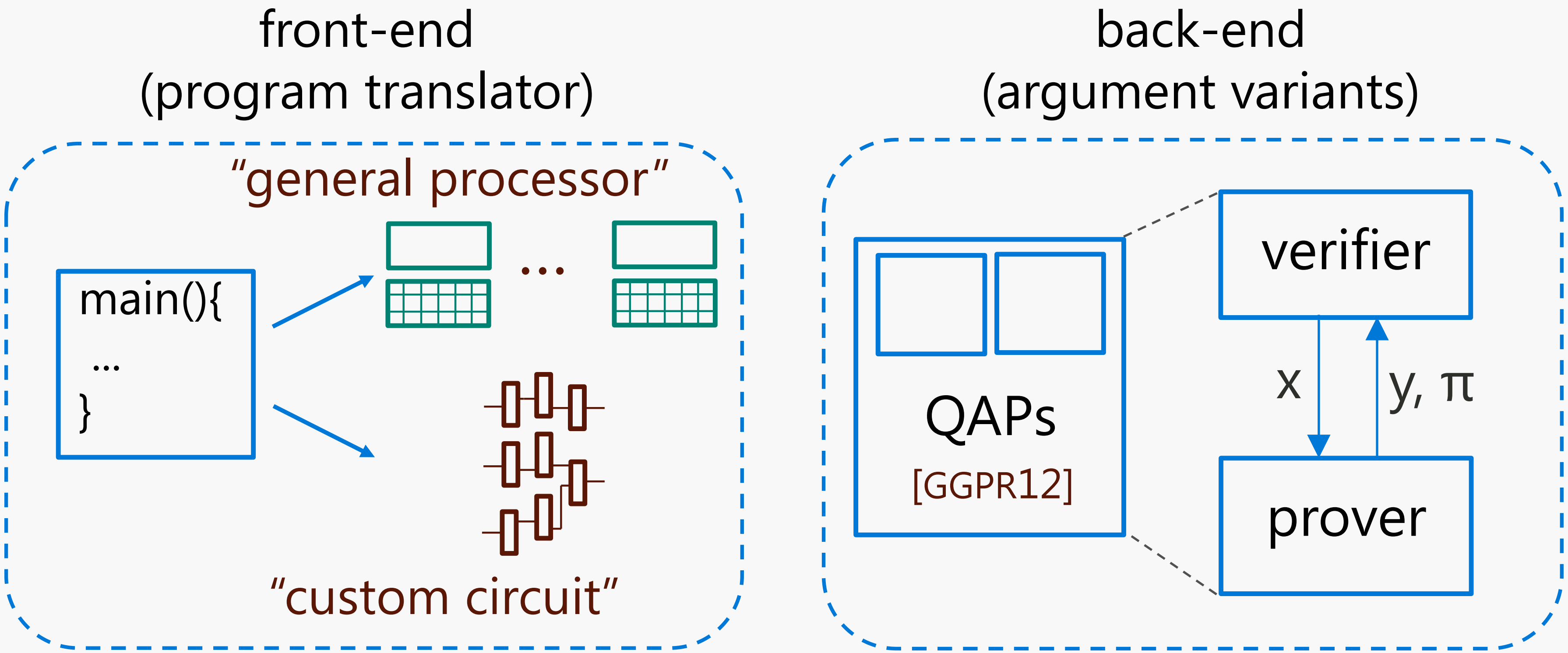
concrete costs	special-purpose	pure	stateful	general loops	function pointers
lower	Thaler CRYPTO13 CMT, TRMP ITCS, Hotcloud12 Pepper, Ginger NDSS12, Security12 Trueset, Zerocash Security14,Oakland15	Zaatar Eurosys13 Pinocchio Oakland13	Geppetto Oakland15 Pantry SOSP13	?	
higher				BCTV Security14 BCGTV CRYPTO13	
highest (still theory)				Proof-carrying data CRYPTO14, Eurocrypt15 Short PCPs Eurocrypt17	

better

Custom circuit

General "processor"

Summary of common framework:



Summary of state of the art implementations

Reality check with a performance evaluation

Next steps

Quick performance study

Back-end: libsnark i.e., BCTV's optimized Pinocchio implementation

Front-ends: implementations or re-implementations of

- Zaatar (Custom circuit) [SBVBPW Eurosys13]
- BCTV (General processor) [Security14]
- Buffet (Custom circuit) [WSRHBW NDSS15]

Landscape of front-ends (again)

applicable computations

concrete costs	special-purpose	pure	stateful	general loops	function pointers
lower	Thaler CRYPTO13		better		
	CMT, TRMP ITCS, Hotcloud12				
	Pepper, Ginger NDSS12, Security12 Trueset, Zerocash Security14, Oakland15	Zaatar Eurosys13 Pinocchio Oakland13	Geppetto Oakland15 Pantry SOSP13	Buffet NDSS15	
					BCTV Security14 BCGTV CRYPTO13
higher					
highest (still theory)				Proof-carrying data CRYPTO14, Eurocrypt15 Short PCPs Eurocrypt17	

Quick performance study

Back-end: libsnark i.e., BCTV's optimized Pinocchio implementation

Front-ends: implementations or re-implementations of:

- Zaatar (Custom circuit) [SBVBPW Eurosys13]
- BCTV (General processor) [Security14]
- Buffet (Custom circuit) [WSRHBW NDSS15]

Evaluation platform: cluster at Texas Advanced Computing Center (TACC)

Each machine runs Linux on an Intel Xeon 2.7 GHz with 32GB of RAM.

(1) What are the verifier's costs?

(2) What are the prover's costs?

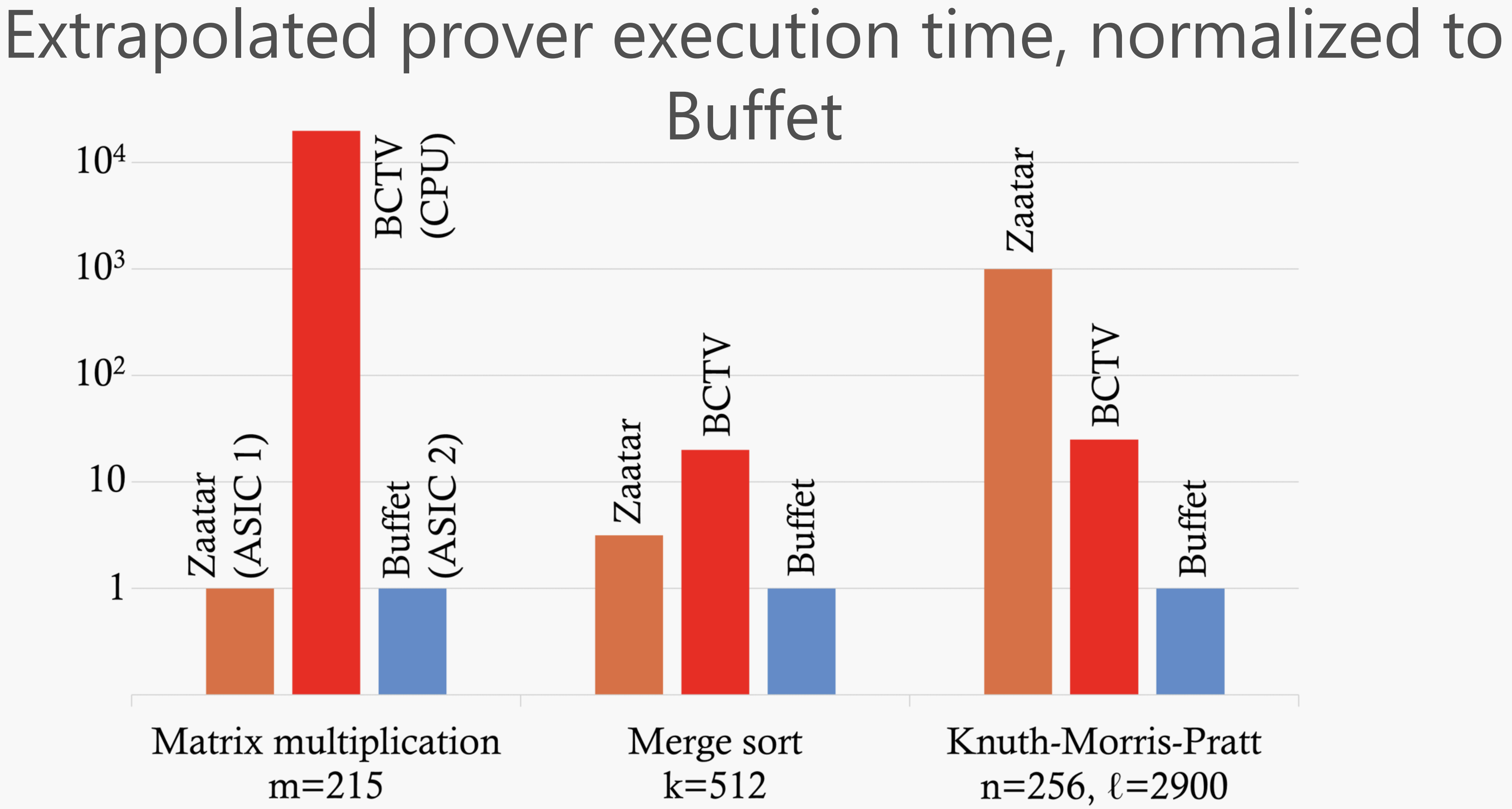
Proof length	288 bytes
V per-instance	6 ms + $(\lvert x\rvert + \lvert y\rvert) \cdot 3$ μs
V pre-processing	$\lvert C\rvert \cdot 180$ μs
P per-instance	$\lvert C\rvert \cdot 60$ μs $+ \lvert C\rvert \log \lvert C\rvert \cdot 0.9$μs
P's memory requirements	$O(\lvert C\rvert \log \lvert C\rvert)$

($|C|$: circuit size)

(3) How do the front-ends compare to each other?

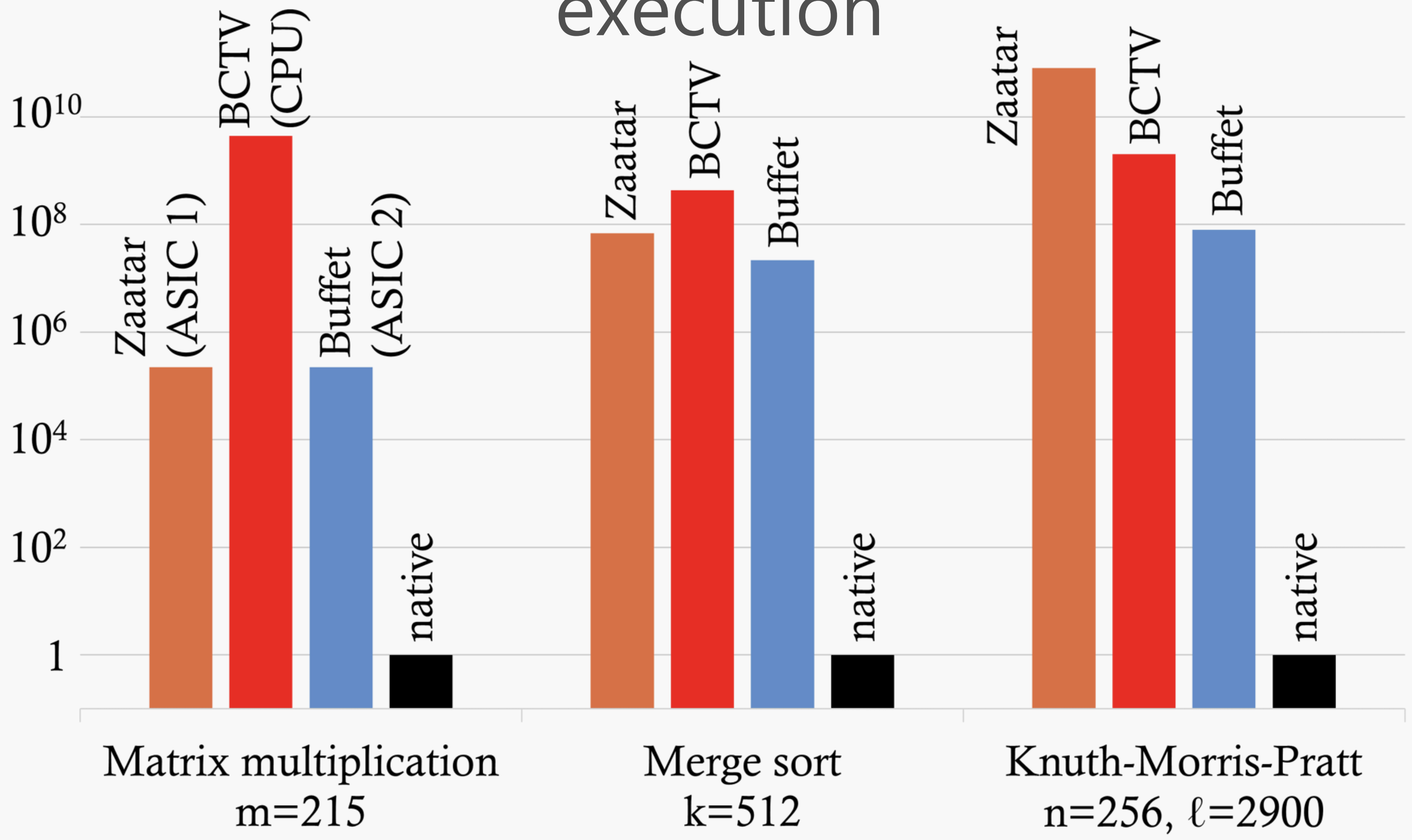
(4) Are the constants good or bad?

How does the prover's cost vary with the choice of front-end?



All of the front-ends have terrible concrete performance

Extrapolated prover execution time, normalized to native execution



Summary of concrete performance

- Front-end: generality brings a concrete price (but better in theory)

- Verifier's "variable costs": genuinely inexpensive
- Prover's computational costs: near-total disaster

- Memory: creates scaling limit for verifier and prover

Where d

Caution!

- Proof generation takes many minutes
- Needs trusted setup
- Prover needs queries that are many GBs

One option: ly execute the computa

- Anonymous credentials: Cinderella [Oakland16]
- Anonymity for Bitcoin: Zerocash [Oakland14]
- Location-private tolling [Security09]: Pantry [SOSP13]

Another option: try to motivate theoretical advances

Summary of state of the art implementations

Reality check with a performance evaluation

Next steps: We need 3-6 orders of magnitude speedup

Wish list (1): front-end

- More efficient reductions from programs to circuits
- Inexpensive floating-point operations (to target domains such as deep learning, machine learning, …)
- Better handling of state

Status quo: systems that handle external state

	Pantry [SOSP13]	ADSNARK [S&P15], Geppetto [S&P15]	Hash first argue later [CCS16]
Technique	CRHF in circuit (folklore)	SNARK already has a CRHF	SNARK already has a CRHF
Generality	Any circuit	Specific	Any circuit
Prover expense	O(k log(|D|))	O(k |D|)	O(k |D|)
Concrete expense	10^6 to 10^8x	10^6 to 10^8x	10^6 to10^8x

- vSQL [S&P17] recently proposed an approach based on polynomial commitments, but it also opens the entire database inside circuit.

- Bottom line: handling state adds additional expense.

Wish list (2): back-end

- Construct short PCPs that are efficient

  Ben-Sasson et al. [EUROCRYPT17] have taken steps toward this, but concrete costs are quite high
- Endow IKO's arguments with more properties or lower costs
  - Reuse the verifier's setup work beyond a batch
  - Make the protocol zero knowledge
- Add zero-knowledge inexpensively to GKR's protocol
- Improve GGPR's QAPs or the cryptography used to query it

Wish list (3): Mission-critical applications

- Verifiable database with support for industrial-grade features: multiple users, concurrency, indexes, etc.

- Screaming performance for the prover e.g., tens of
th

Lots of other ideas needed; we don’t know what they are!

- Why? DBs process financial transactions worth trillions of dollars. Connections with emerging distributed ledgers.

Conclusions and takeaways

- Exciting area with good news and bad news
  - **Lots of progress, but …**
  - **… extreme expense in general-purpose systems**
- Overheads rooted in QAPs and circuit representation
- Theoretical breakthroughs are needed

- Incentive: the potential is huge, especially with emerging distributed ledgers

(http://www.pepper-project.org/)