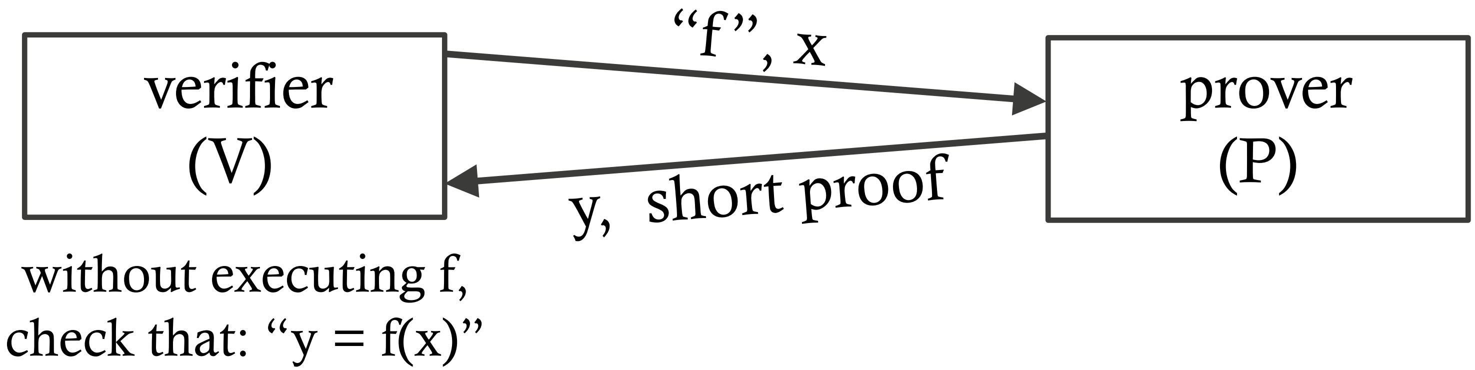
Introduction and overview of verifiable computation

(≈ delegation of computation
≈ succinct arguments
≈ execution integrity)

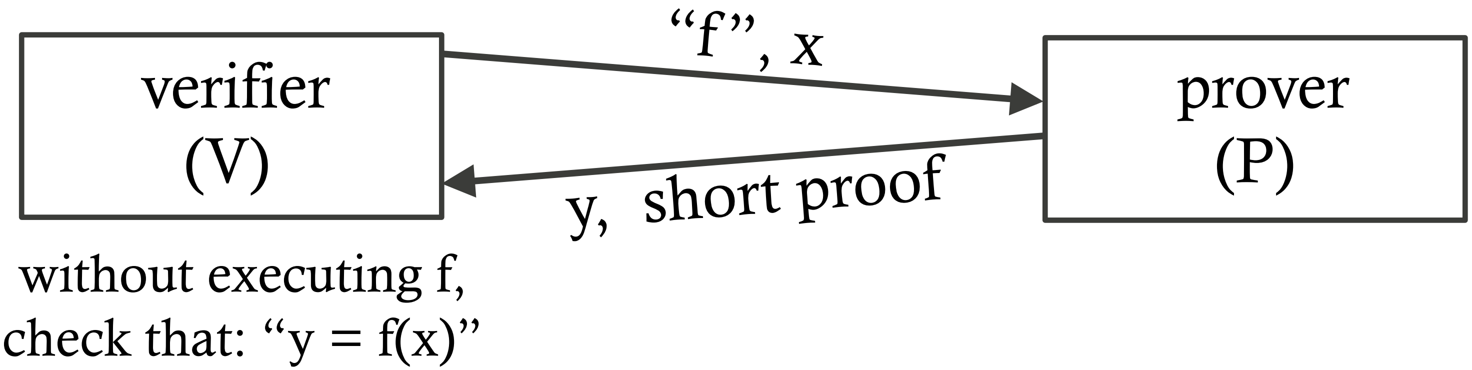
Bar-Ilan Winter School on Verifiable Computation
Class 1
January 4, 2016

Michael Walfish
Dept. of Computer Science, Courant Institute, NYU



Classic and fundamental problem

- Cloud computing (consider large distributed jobs)
- Information retrieval (consider a query against a remote database)
- Hardware supply chain (consider potentially adversarial chips)
- Generalizes to verifying assertions
- Many other applications

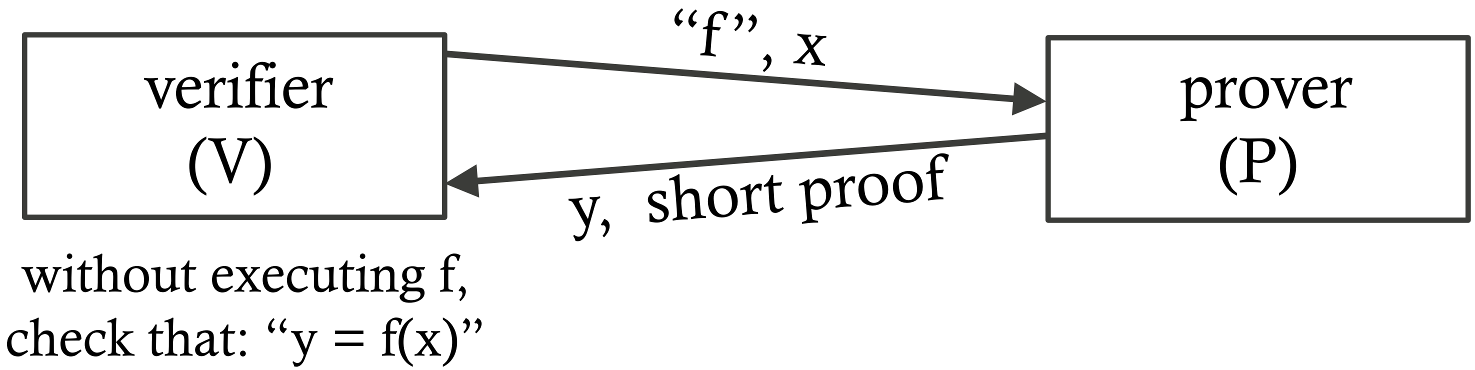


Classic and fundamental problem

- Many applications (cloud computing, information retrieval, untrusted hardware supply chain, etc.). Generalizes to verifying assertions.

Many variants of the setup

- Proof delivered over rounds of interaction
- More general claim: "there exists a w such that y = f(x,w)"
  - ... and furthermore P "knows" w
  - ... and furthermore P can keep w private
- Different assumptions (unconditional vs. standard vs. funky)
- V cannot access all of its input



Classic and fundamental problem

- Many applications (cloud computing, information retrieval, untrusted hardware supply chain, etc.). Generalizes to verifying assertions.

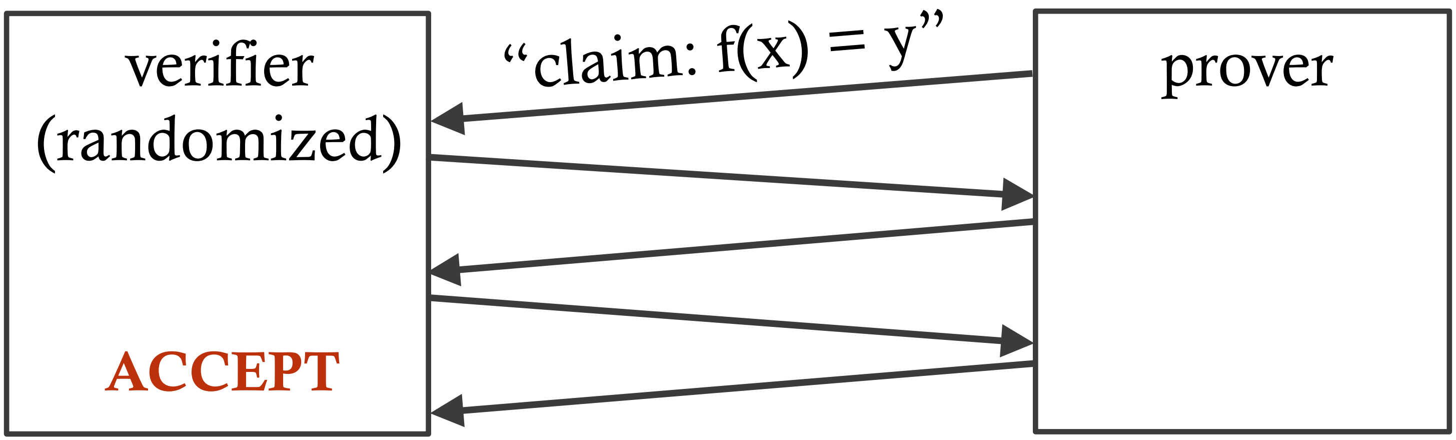
Many variants of the setup

- Commonality: V gets assurance that P performed a task as directed, without redoing P's work and without access to P's resources or inputs.

Note: program correctness is complementary

- Program correctness establishes that f is consistent with a specification. In our context, f is a (possibly buggy) given and is the directive for P.

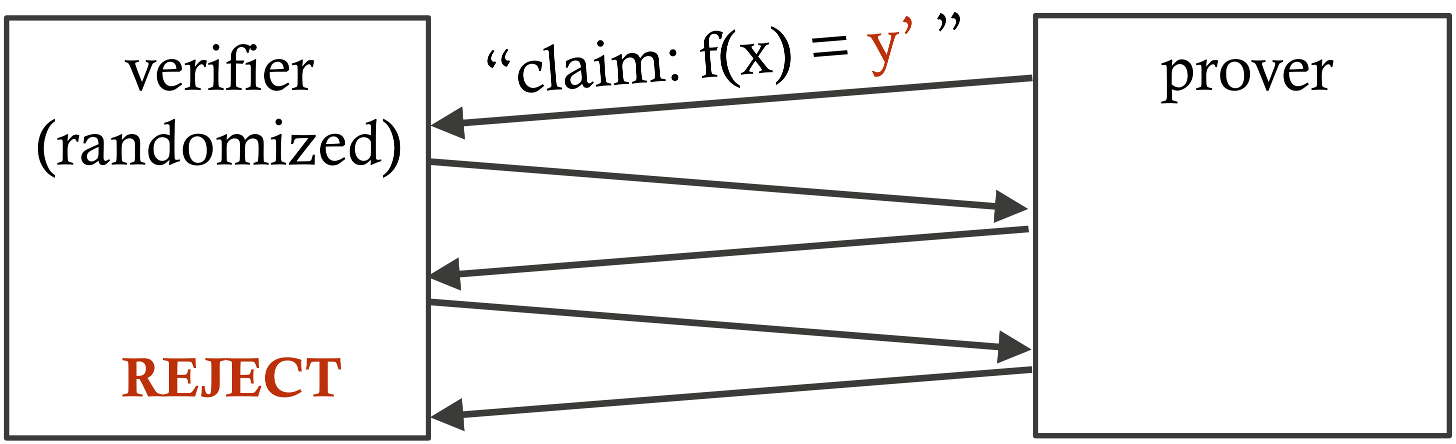
Many of the variants are addressable in theory, with probabilistic proof protocols



Citations here and throughout connect to the references at the end of the slides.

GMR85
BCC86
BFLS91
FGLSS91
Kilian92
ALMSS92
AS92
Micali94
BG02
GOS06
IKO07
GKR08
KR09
GGP10
Groth10
GLR11
Lipmaa11
BCCT12
GGPR13
BCCT13
KRR14
...

Many of the variants are addressable in theory, with probabilistic proof protocols



Are probabilistic proof protocols practical?

GMR85
BCC86
BFLS91
FGLSS91
Kilian92
ALMSS92
AS92
Micali94
BG02
GOS06
IKO07
GKR08
KR09
GGP10
Groth10
GLR11
Lipmaa11
BCCT12
GGPR13
BCCT13
KRR14
…

Below is a list of published implementations of probabilistic proofs. See [WB15] for a partial survey.

Good news:

- Running code; cost reductions of 10^{20} vs. theory
- Compilers from C to verifiable computations
- Concretely efficient verifiers

Equivocal news:

- Small computations, extreme expense, etc.
- Useful only for special-purpose applications

So, lots of work left … and high payoff:

this is a good opportunity for you!

SBW11
CMT12
SMBW12
TRMP12
SVPBBW12
SBVBPW13
VSBW13
PGHR13
Thaler13
BCGTV13
BFRSBW13
BFR13
DFKP13
BCTV14a
BCTV14b
BCGGMTV14
FL14
KPPSST14
FGP14
WSRHBW15
BBFR15
CFHKKNPZ15
CTV15
KZMQCPPsS15
WHGsW15

Rest of this session

(1) Landscape, history, and synopsis of the area

(2) Syllabus (for the 10 classes on verifiable computation)

(3) Technical preliminaries

Landscape: broad approaches to verifiable computation

A. Make usage assumptions

- replication [MR97, CL99, CRR11]
- attestation [PMP11, SLSPDK05], trusted hardware [CT10, SSW10]
- auditing [MWR99, HKD07]

B. Restrict the class of computations

[Freivalds77, GM01, Sion05, MSG07, KSC09, BGV11, BF11, BZF11, FG12, …]

C. Strive for generality

A brief history of verifiable computation via probabilistic proofs

- Interactive Proofs [GMR85], Arthur-Merlin [Babai85]
- PCPs [BFLS91]

> "In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware."
>
> —Babai, Fortnow, Levin, and Szegedy, Checking Computations in Polylogarithmic Time, 1991

A brief history of verifiable computation via probabilistic proofs

- Interactive Proofs [GMR85], Arthur-Merlin [Babai85]
- PCPs [BFLS91] ("a single reliable PC can monitor…")
- PCP theorem [ALMSS92, AS92]
- Efficient arguments [Kilian92]
- CS proofs [Micali94]

"we aim at obtaining certificates ensuring that no error has occurred in a *given* execution of a *given* algorithm on a *given* input...This question is quite crucial whenever we are confident in the design of a given algorithm ... but less so in the physical computer that runs it."

—Micali, Computationally Sound Proofs, 2000

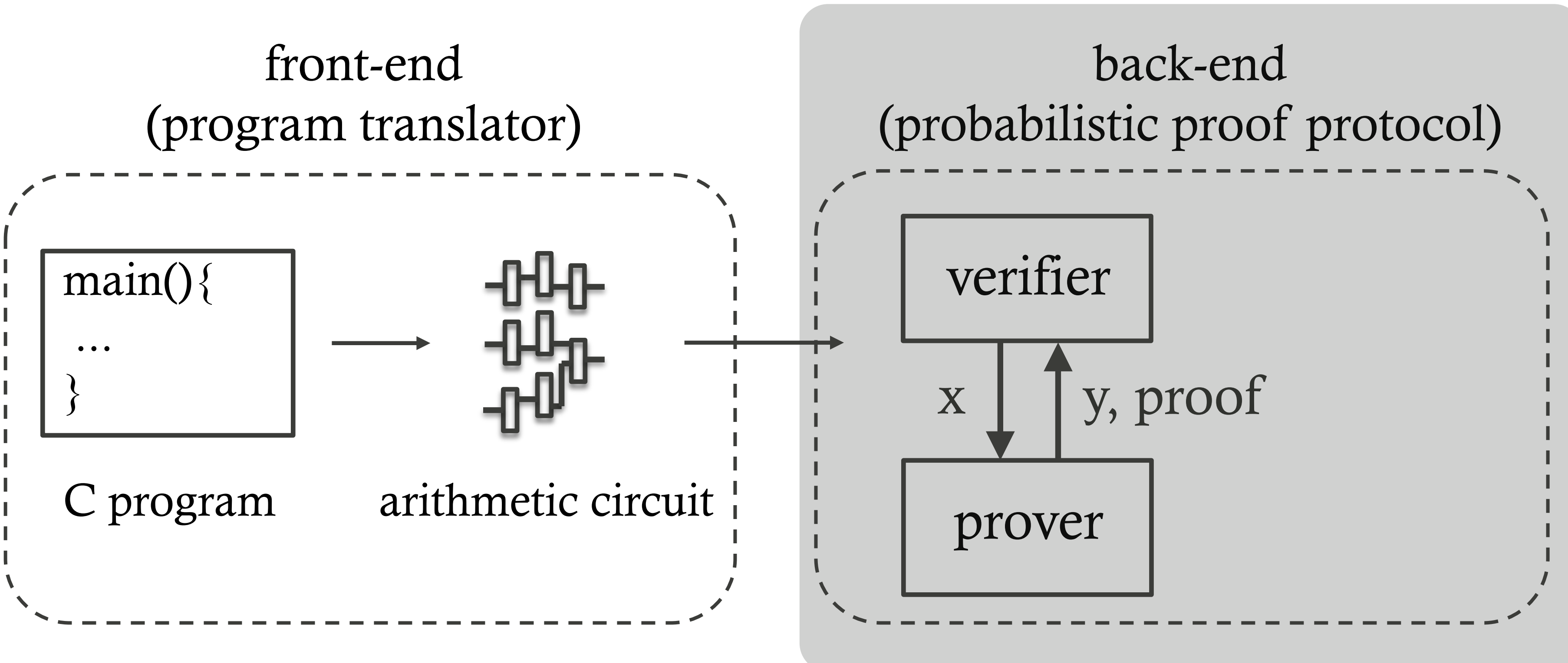
A brief history of verifiable computation via probabilistic proofs

- Interactive Proofs [GMR85], Arthur-Merlin [Babai85]
- PCPs [BFLS91] ("a single reliable PC can monitor…")
- PCP theorem [ALMSS92, AS92]
- Efficient arguments [Kilian92]
- CS proofs [Micali94] ("certified computation")

- Interactive proof with polynomial prover [GKR08]
- Efficient argument with simple PCP [IKO07]
- Non-interactive verifiable computation [GGP10] (coins "VC")

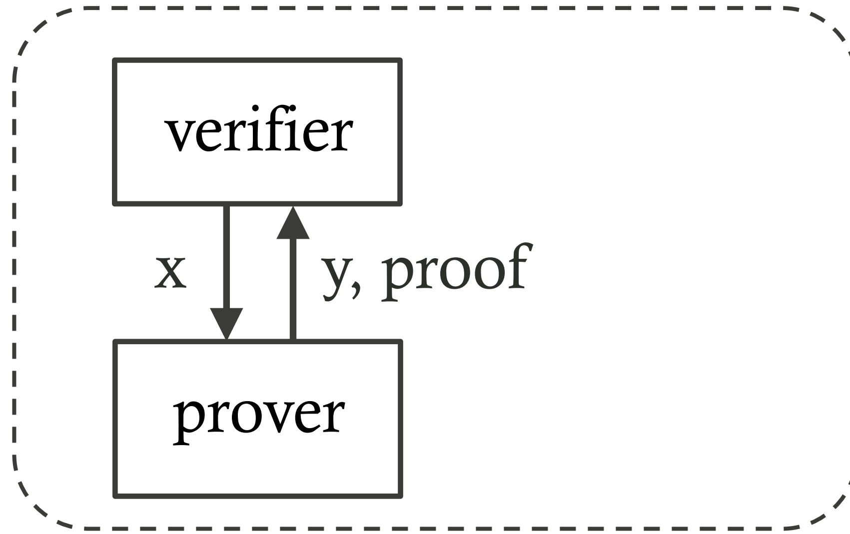
- Challenges to the view that "this is theory-only" (2011–) [WB15]
- Theoretical innovation ongoing: SNARG/SNARK [GW11, Groth10, Lipmaa12, GGPR12, BCCT13, BCCGLRT14], 2-msg delegation [KRR14], …

Synopsis of the research area



- what circuits does it handle?
- what assumptions are needed?
- what are its properties?
- what is the number of messages?
- what are the costs?
- what costs can be amortized?
- what are the mechanics?

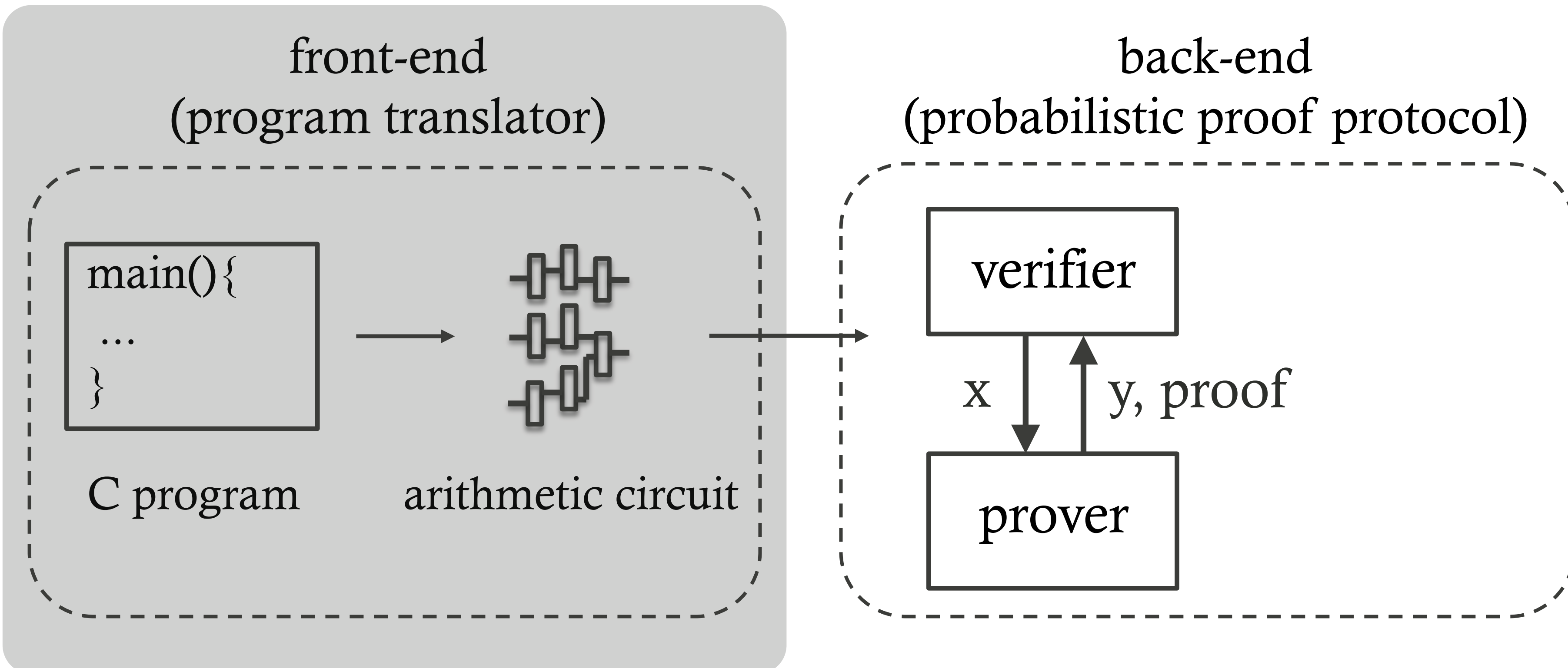
back-end
(probabilistic proof protocol)



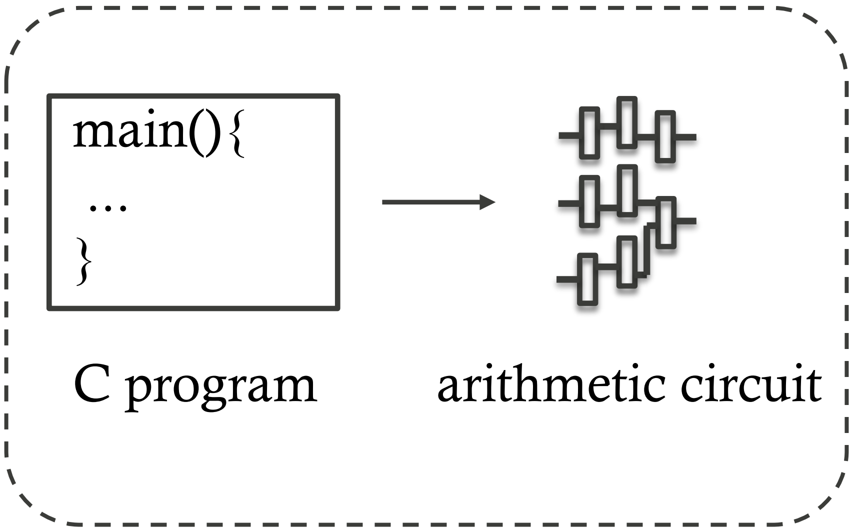
interactive proof

interactive argument

non-interactive argument
(CS proof, SNARG, SNARK)


front-end
(program translator)
main(){
...
}
C program
arithmetic circuit
back-end
(probabilistic proof protocol)
verifier
x
y, proof
prover

front-end
(program translator)



- how expressive is it?
- what is programming like?
- how does translation work?
- what are the costs of different program structures?
- how can programs refer to external state?

circuits with repeated structure
circuits without repeated structure
circuits w/ non-deterministic input
} "ASIC"

"universal" circuits } "CPU"

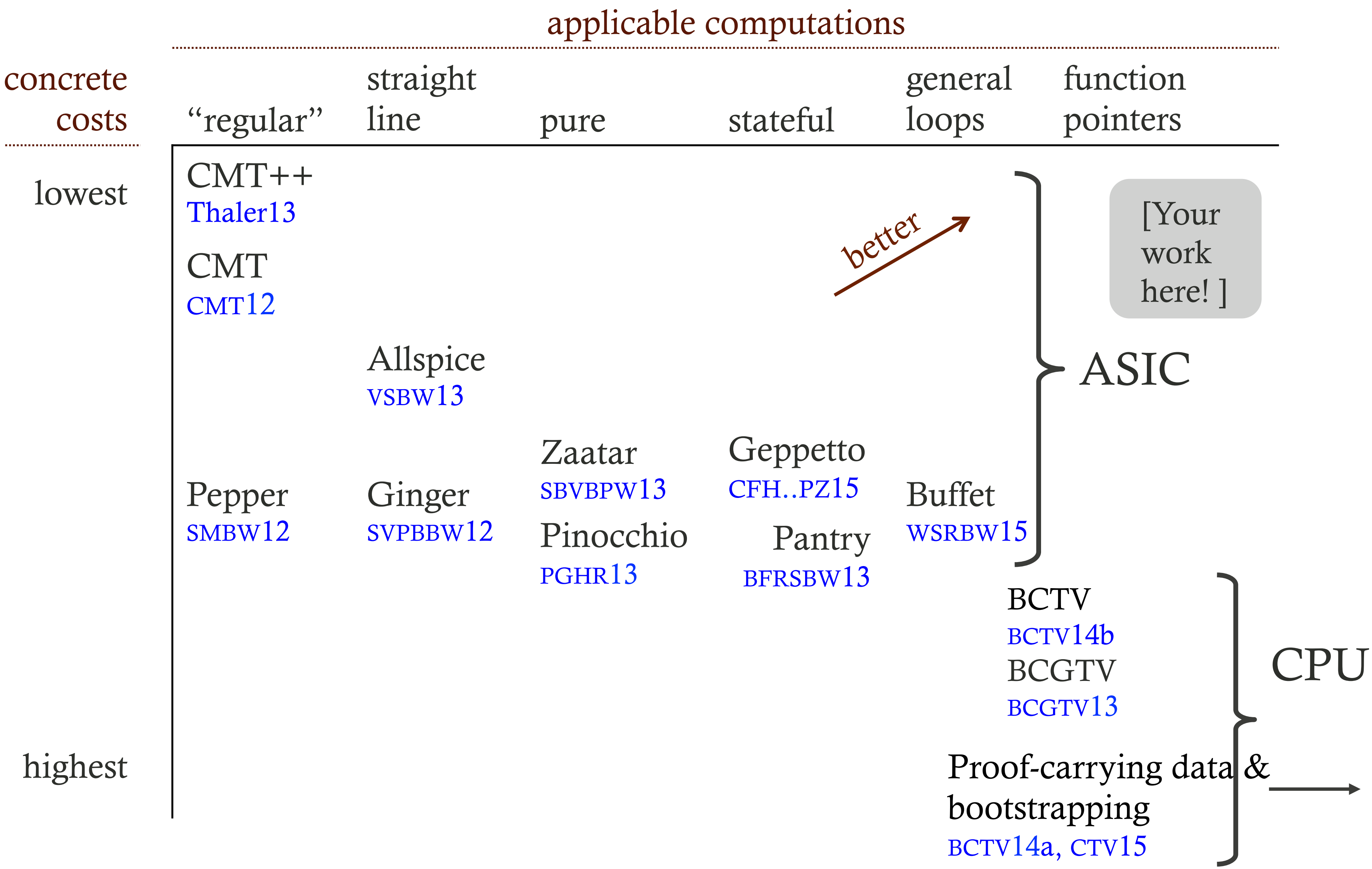
A key trade-off is performance versus expressiveness

applicable computations

concrete costs	"regular"	straight line	pure	stateful	general loops	function pointers	
lowest	CMT++ Thaler13						interactive proofs (IPs)
	CMT CMT12						interactive proofs (IPs)
		Allspice VSBW13					interactive proofs (IPs)
	Pepper SMBW12	Ginger SVPBBW12	Zaatar SBVBPW13 Pinocchio PGHR13	Geppetto CFH..PZ15 Pantry BFRSBW13	Buffet WSRBW15		args.
						BCTV BCTV14b BCGTV BCGTV13	args.
highest					Proof-carrying data & bootstrapping BCTV14a, CTV15		args.

better

A key trade-off is performance versus expressiveness



The area is interdisciplinary:

- We care about interesting theory and concrete costs
- The area blends crypto, complexity theory, PL, systems

Lots of open problems and questions

- Unconditionally secure delegation for all of PSPACE (YTK \$100)
- 2-msg delegation for $\mathcal{NP}$ with standard assumptions (YTK)
- Publicly-verif. 2-msg delegation for $\mathcal{P}$ with std. assumptions (YTK)
- Zero knowledge with standard assumptions that is inexpensive in practice
- More efficient reductions from programs to circuits
- More efficient encodings of execution traces
- Probabilistic proof protocols that do not require circuits
- Avoiding preprocessing/amortization in a way that is inexpensive in practice
- Special-purpose algorithms for outsourcing pieces of computations, which integrate with circuit verification

(1) Landscape, history, and synopsis of the area

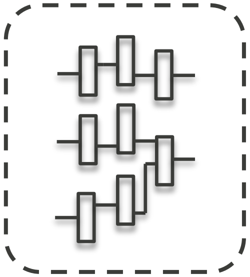
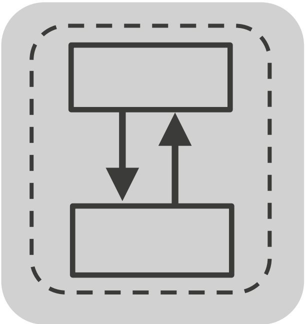
(2) Syllabus (for the 10 classes on verifiable computation)

(3) Technical preliminaries

Our goal: motivate and equip you to do research in this area

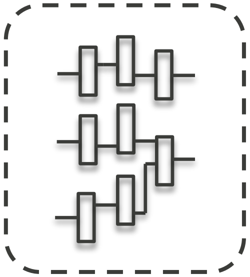
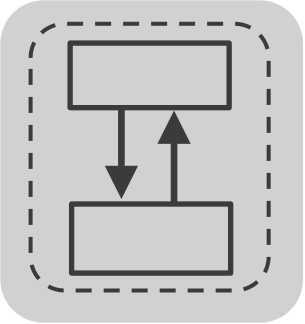
How:

- Teach you some of the building blocks
- Expose you to the key results
- Provide you with pointers into the literature

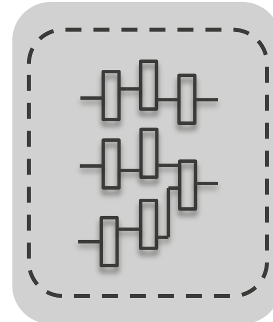
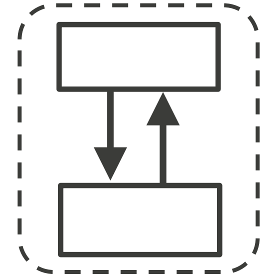
- Class 2: Statistically sound delegation (YTK)
  - History
  - Sum-check protocol, low-degree extensions
  - Unconditionally secure delegation for low depth circuits

- Classes 3 and 4: Computationally sound delegation (YTK)
  - History of arguments and CS proofs
  - PCP + hash paradigm, Fiat-Shamir heuristic
  - The space of assumptions
  - 2-msg delegation for computations in $\mathcal{P}$ (std. assumptions) …
  - … and for "long input" computations

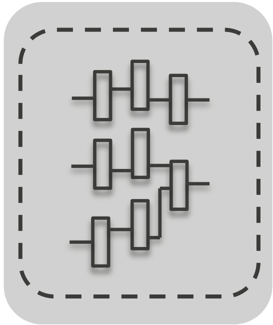
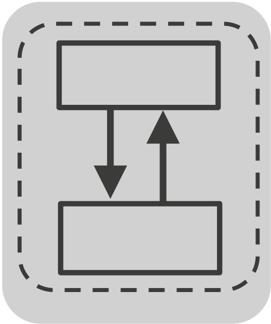
 

- Class 5: Interactive arguments with preprocessing (MW)
  - Linear PCPs
  - Interactive arguments via linear PCPs
  - The role of QAPs

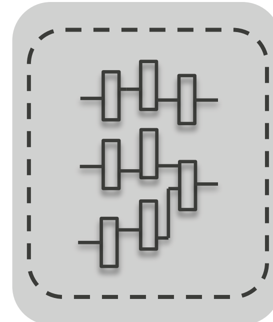
- Class 6: Non-interactive arguments with preprocessing (ET)
  - SNARGs and (zk-)SNARKs based on linear PCPs
  - Details of QAPs
  - Refinements of QAPs

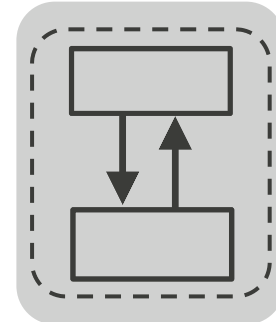
 

- Class 7: Program representations (MW)
  - Arithmetization: from programs to circuits ("ASIC approach")
  - Data-dependent control flow
  - Expressiveness versus amortization versus performance

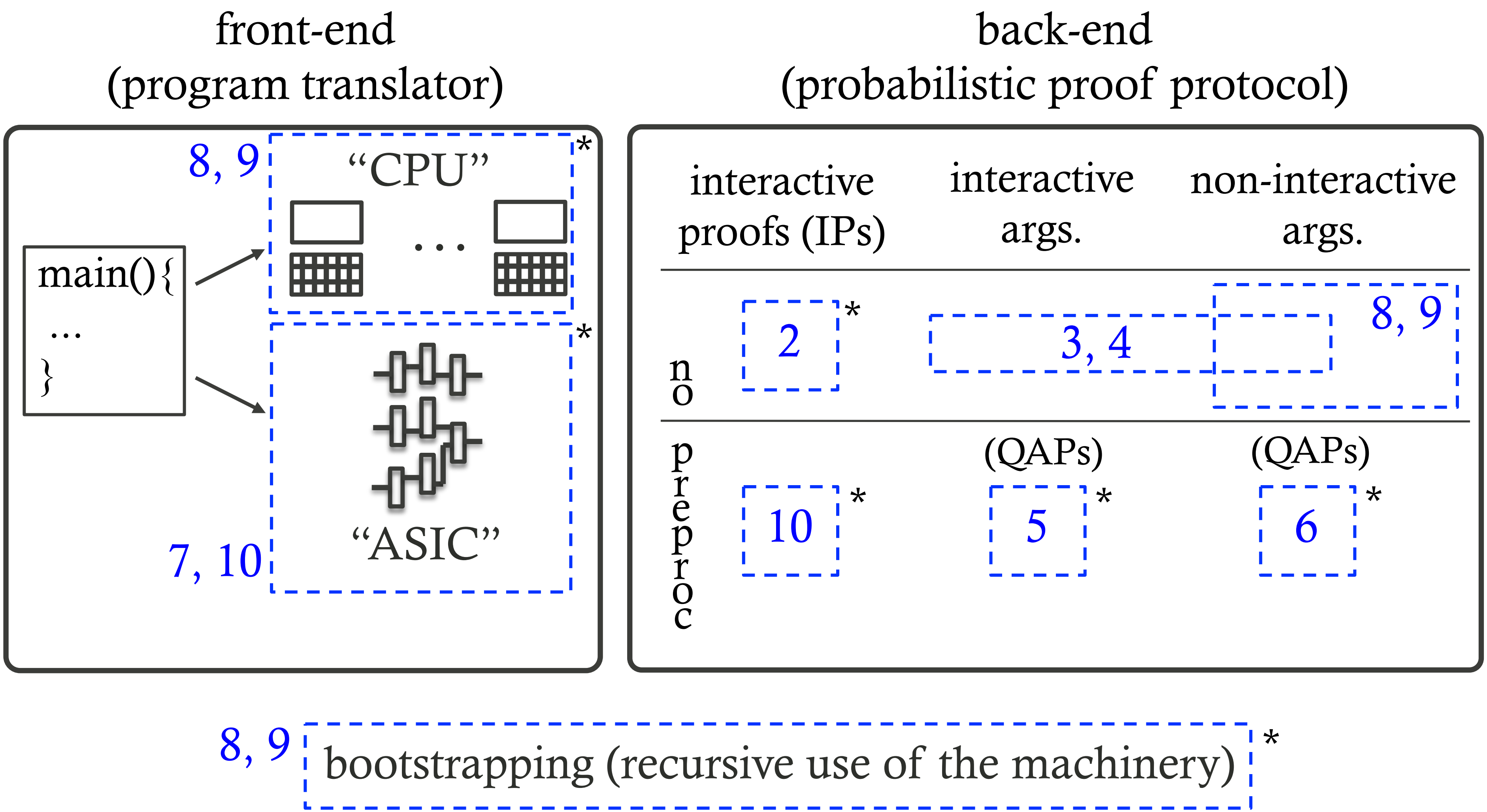
- Classes 8 and 9: TBA (ET). Possible topics include:
  - Application of "ASIC approach": Zerocash [BCGGMTV14]
  - "CPU approach" to circuits: TinyRAM [BCGTV13, BCTV14b]
  - Permutation networks for RAM computations [BCGT13, BCGTV13, BCTV14b]
  - Bootstrapping SNARKs [BCCT13] by composing QAPs, TinyRAM, and elliptic curve cycles [BCTV14a, CTV15]
  - SNARKs without preprocessing, using short PCPs [BCCT12, BCCGLRT14]
  - Progress in ongoing work implementing short PCPs





- Class 10: Additional applications and summary (MW)
  - External state
  - MapReduce, face-matching, regression analysis, etc.
  - Implementations of IPs (time permitting)
  - Wrap-up

Classes at a glance (numbers in blue refer to class number)



* Indicates that the mechanism has been implemented

(1) Landscape, history, and synopsis of the area

(2) Syllabus (for the 10 classes on verifiable computation)

(3) Technical preliminaries

How are probabilistic proofs defined?

Completeness:

Soundness:

Efficiency:

Variants: computational soundness, non-deterministic languages, proof of knowledge, zero knowledge.

How are probabilistic proofs defined?

There are many definitions and variants; below is the general form. For details, consult a text ([AB09, Goldreich07, Micali94, BG02] are all extremely lucid). A probabilistic proof for a language L is an interacting verifier V_L (which is PPT) and prover P_L (whose power varies depending on the definition). Let (V, P)(a) denote the interaction between V and P on instance a. If (V, P)(a) = 1, V is said to "accept" the interaction. The interaction must meet:

Completeness:

If $a \in L$, $\Pr\{(V_L, P_L)(a) = 1\} = 1$.

The probability is taken over V's random choices.

Soundness:

If $a \notin L$, then $\forall$ P', $\Pr\{(V_L, P')(a) = 1\} < \varepsilon$, for some fixed, constant ε.

The probability is again over V's random choices.

Efficiency:

The *honest* P (that is, P_L) should have running time that is polynomial (and ideally linear or quasilinear) in the time to compute or decide L (as noted earlier, the assumed power of a *dishonest* P depends on the kind of probabilistic proof). V's running time is ideally constant or logarithmic in the time to compute or decide L; same with the communication complexity.

Variants: computational soundness, non-deterministic languages, proof of knowledge, zero knowledge.

What language should we use for "correct program execution"?

- Boolean circuit satisfiability

- Arithmetic circuit satisfiability

- Non-deterministic (Boolean, arith.) circuit satisfiability

"Satisfiability" enters because there are implicit constraints.
Sometimes it is easier to work with constraints explicitly.

What language should we use for “correct program execution”?

- Boolean circuit satisfiability

  We use this term to refer to the language of triples (C, x, y) where a Boolean circuit C, if given input x, produces output y. This is slightly non-standard, but it matches the problem setup in delegation.

- Arithmetic circuit satisfiability

  Similar to prior one, but now: the circuit is over a large finite field, the wires are interpreted as field elements, and the gates are interpreted as field operations (add, multiply).

- Non-deterministic (Boolean, arith.) circuit satisfiability

  Now we imagine that the circuit takes some unconstrained input (label it W), and this language is all triples (C, x, y) for which there exists some W=w such that C(x,w) = y.

“Satisfiability” enters because there are implicit constraints. Sometimes it is easier to work with constraints explicitly.

A convenient language: arithmetic constraint satisfiability

- System of equations in finite field $\mathbb{F}$.

- A computation f is equivalent to constraints C if:

increment-by-one

```
f(X) {
  Y = X + 1;
  return Y;
}
```

A convenient language: arithmetic constraint satisfiability

- System of equations in finite field $\mathbb{F}$.

- A computation f is equivalent to constraints C if:

  C is constraints over variables (X, Y, Z) and field $\mathbb{F}$ s.t.
  (det case) $\forall x,y$: $y=f(x) \Leftrightarrow C(X=x, Y=y)$ is satisfiable
  (non-det. case) $\forall x,y$: $(\exists\ w$ s.t. $y=f(x,w)) \Leftrightarrow C(X=x, Y=y)$ is satisfiable
  Terminology: constraints C are said to be an arithmetization of the computation f.

increment-by-one

```
f(X) {
  Y = X + 1;
  return Y;
}
```

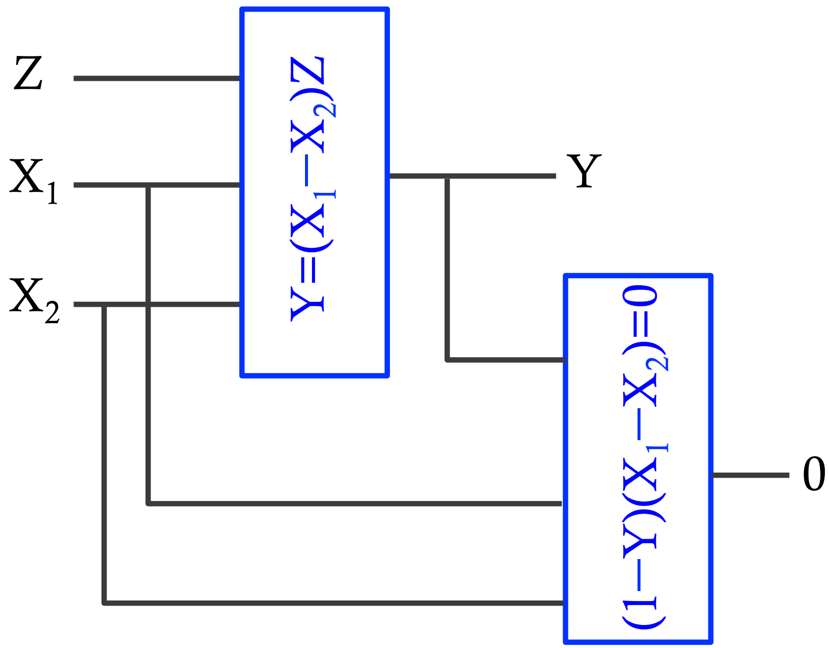
[equivalent]

$$\left\{ \begin{array}{l} 0 = Z - X, \\ 0 = Z - Y + 1 \end{array} \right\}$$

QuadConstraint$_{\mathbb{F}}$

- Degree-2 constraints over finite field $\mathbb{F}$

- What do the constraints/gates below represent?



Summary

- This is an exciting inter-disciplinary area:
  - Addresses a fundamental problem, using deep theory
  - There is still lots of work to be done …
  - …. but the potential is large (goes far beyond the cloud!)
- Central technical notions:
  - probabilistic proofs, circuits (constraints), program translators
- Many tradeoffs, properties, axes, facets
  - Ideally, we will help you understand them
  - Ideally, you will help improve them!

Acknowledgment: my collaborators in this research area

Andrew J. Blumberg (UT), Benjamin Braun (now at Stanford), Ariel Feldman (UChicago), Siddharth Garg (NYU), Max Howald (Cooper Union, NYU), Richard McPherson, Nikhil Panpalia (now at Amazon), Bryan Parno (MSR), Zuocheng Ren, Srinath Setty (now at MSR), abhi shelat (UVA), Justin Thaler (Yahoo Research), Victor Vu (now at Sandia), and Riad Wahby (now at Stanford).

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

References

[AB09] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998. Prelim. version FOCS 1992.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998. Prelim. version FOCS 1992.

[Babai85] L. Babai. Trading group theory for randomness. In *STOC*, May 1985.

[BBFR15] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *IEEE Symposium on Security and Privacy*, May 2015.

[BCC86] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and Systems Sciences*, 37(2):156–189, October 1988. Prelim. versions: several papers in CRYPTO and FOCS 1986.

[BCCGLRT14] N. Bitansky, R. Canetti, A. Chiesa, S. Goldwasser, H. Lin, A. Rubinstein, and E. Tromer. The hunting of the SNARK. Cryptology ePrint Archive, Report 2014/580. 2014.

[BCCT12] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, January 2012.

[BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *STOC*, June 2013.

[BCGGMTV14] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Decentralized anonymous payments from Bitcoin. In *IEEE Symposium on Security and Privacy*, May 2014.

[BCGT13] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. On the concrete-efficiency threshold of probabilistically-checkable proofs. In *STOC*, June 2013.

[BCGTV13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO*, August 2013.

[BCIOP13] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. Succinct non-interactive arguments via linear interactive proofs. In *IACR TCC*, March 2013.

[BCTV14a] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO*, August 2014.

[BCTV14b] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security Symposium*, August 2014.

[BEGKN91] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. In *FOCS*, October 1991.

[BF11] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *Eurocrypt*, May 2011, pages 149–168.

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *STOC*, May 1991.

[BFR13] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *ACM CCS*, November 2013.

[BFRSBW13] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish. Verifying computations with state. In *SOSP*, November 2013.

[BG02] B. Barak and O. Goldreich. Universal arguments and their applications. *SIAM Journal on Computing*, 38(5):1661–1694, 2008. Prelim. version CCC 2002.

[BGHSV05] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Short PCPs verifiable in polylogarithmic time. In *Conference on Computational Complexity (CCC)*, 2005.

[BGHSV06] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, December 2006.

[BGV11] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, August 2011, pages 111–131.

[Braun12] B. Braun. Compiling computations to constraints for verified computation. UT Austin Honors thesis HR-12-10. December 2012.

[BS08] E. Ben-Sasson and M. Sudan. Short PCPs with polylog query complexity. *SIAM Journal on Computing*, 38(2):551–607, May 2008.

[BZF11] M. Blanton, Y. Zhang, and K. Frikken. Secure and verifiable outsourcing of large-scale biometric computations. In *IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT)*, October 2011.

[CFHKKNPZ15] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: versatile verifiable computation. In *IEEE Symposium on Security and Privacy*, May 2015.

[CL99] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002. Prelim. versions OSDI 1999, OSDI 2000.

[CMT12] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, January 2012.

[CRR11] R. Canetti, B. Riva, and G. Rothblum. Practical delegation of computation using multiple servers. In *ACM CCS*, October 2011.

[CT10] A. Chiesa and E. Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS*, 2010.

[CTV15] A. Chiesa, E. Tromer, and M. Virza. Cluster computing in zero knowledge. In *Eurocrypt*, April 2015, pages 371–403.

[DFKP13] G. Danezis, C. Fournet, M. Kohlweiss, and B. Parno. Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In *Workshop on Language Support for Privacy-enhancing Technologies*, November 2013.

[Din07] I. Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3), June 2007.

[FG12] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *ACM CCS*, May 2012, pages 501–512.

[FGLSS91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, March 1996. Prelim. version FOCS 1991.

[FGP14] D. Fiore, R. Gennaro, and V. Pastro. Efficiently verifiable computation on encrypted data. In *ACM CCS*, 2014.

[FL14] M. Fredrikson and B. Livshits. ZØ: an optimizing distributing zero-knowledge compiler. In *USENIX Security Symposium*, August 2014.

[Freivalds77] R. Freivalds. Probabilistic machines can use less running time. In *Proceedings of the IFIP Congress*, 1977, pages 839–842.

[GF95] A. M. Ghuloum and A. L. Fisher. Flattening and parallelizing irregular, recurrent loop nests. In *ACM PPoPP*, July 1995.

[GGP10] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO*, August 2010.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Eurocrypt*, 2013.

[GKR08] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM*, 62(4):27:1–27:64, August 2015. Prelim. version STOC 2008.

[GLR11] S. Goldwasser, H. Lin, and A. Rubinstein. Delegation of computation without rejection problem from designated verifier CS-proofs. Cryptology ePrint Archive, Report 2011/456. 2011.

[GM01] P. Golle and I. Mironov. Uncheatable distributed computations. In *RSA Conference*, April 2001, pages 425–440.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. Prelim. version STOC 1985.

[Goldreich07] O. Goldreich. Probabilistic proof systems – a primer. *Foundations and trends in theoretical computer science*, 3(1):1–91, 2007.

[GOS06] J. Groth, R. Ostrovsky, and A. Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM*, 59(3):11:1–11:35, June 2012. Prelim. versions CRYPTO 2006, Eurocrypt 2006.

[Groth10] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Asiacrypt*, 2010.

[GW11] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, June 2011.

[HKD07] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: practical accountability for distributed systems. In *SOSP*, October 2007, pages 175–188.

[IKO07] Y. Ishai, E. Kushilevitz, and R. Ostrovsky. Efficient arguments without short PCPs. In *Conference on Computational Complexity (CCC)*, 2007.

[Kilian92] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, May 1992.

[Knijnenburg98] P. M. W. Knijnenburg. Flattening: VLIW code generation for imperfectly nested loops. In *CPC98*, June 1998.

[KNP05] A. Kejariwal, A. Nicolau, and C. D. Polychronopoulos. Enhanced loop coalescing: a compiler technique for transforming non-uniform iteration spaces. In *ISHPC05/ALPS06*, September 2005.

[KP15] Y. T. Kalai and O. Paneth. Delegating RAM computations. Cryptology ePrint Archive, Report 2015/957. 2015.

[KPPSST14] A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos. TRUESET: faster verifiable set computations. In *USENIX Security Symposium*, August 2014.

[KR09] Y. T. Kalai and R. Raz. Probabilistically checkable arguments. In *CRYPTO*, 2009.

[KRR14] Y. T. Kalai, R. Raz, and R. Rothblum. How to delegate computations: the power of no-signaling proofs. In *STOC*, 2014.

[KSC09] G. O. Karame, M. Strasser, and S. Čapkun. Secure remote execution of sequential computations. In *International Conference on Information and Communications Security*, December 2009.

[KZMQCPPsS15] A. Kosba, Z. Zhao, A. Miller, Y. Qian, H. Chan, C. Papamanthou, R. Pass, abhi shelat, and E. Shi. How to use SNARKs in universally composable protocols. Cryptology ePrint Archive, Report 2015/1093. 2015.

[Lipmaa11] H. Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *IACR TCC*, 2011.

[Meir12] O. Meir. Combinatorial PCPs with short proofs. In *Conference on Computational Complexity (CCC)*, 2012.

[Merkle87] R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, August 1987.

[Micali94] S. Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000. Prelim. version FOCS 1994.

[MR97] D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, October 1998. Prelim. version STOC 1997.

[MSG07] N. Michalakis, R. Soulé, and R. Grimm. Ensuring content integrity for untrusted peer-to-peer content distribution networks. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[MWR99] F. Monrose, P. Wycko, and A. D. Rubin. Distributed execution with remote audit. In *Network and Distributed System Security Symposium (NDSS)*, February 1999.

[PGHR13] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, May 2013.

[PMP11] B. Parno, J. M. McCune, and A. Perrig. *Bootstrapping trust in modern computers*. Springer, 2011.

[Polychron87] C. D. Polychronopoulos. Loop coalescing: a compiler transformation for parallel machines. In *ICPP*, August 1987.

[SBVBPW13] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Eurosys*, April 2013.

[SBW11] S. Setty, A. J. Blumberg, and M. Walfish. Toward practical and unconditional verification of remote computations. In *Workshop on Hot Topics in Operating Systems (HotOS)*, May 2011.

[Sion05] R. Sion. Query execution assurance for outsourced databases. In *International Conference on Very Large Databases (VLDB)*, August 2005, pages 601–612.

[SLSPDK05] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: verifying integrity and guaranteeing execution of code on legacy platforms. In *SOSP*, October 2005.

[SMBW12] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *Network and Distributed System Security Symposium (NDSS)*, February 2012.

[SSW10] A.-R. Sadeghi, T. Schneider, and M. Winandy. Token-based cloud computing: secure outsourcing of data and arbitrary computations with lower latency. In *TRUST*, June 2010.

[SVPBBW12] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security Symposium*, August 2012.

[Thaler13] J. Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO*, August 2013.

[TRMP12] J. Thaler, M. Roberts, M. Mitzenmacher, and H. Pfister. Verifiable computation with massively parallel interactive proofs. In *USENIX HotCloud Workshop*, June 2012.

[VSBW13] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. In *IEEE Symposium on Security and Privacy*, May 2013.

[WB15] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them: from theoretical possibility to near practicality. *Communications of the ACM*, 58(2):74–84, February 2015.

[WHGsW15] R. Wahby, M. Howald, S. Garg, abhi shelat, and M. Walfish. Verifiable ASICs. Preprint. December 2015.

[WSRBW15] R. S. Wahby, S. Setty, Z. Ren, A. J. Blumberg, and M. Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *Network and Distributed System Security Symposium (NDSS)*, February 2015.

[YCFVEEGH08] B. Ylvisaker, A. Carroll, S. Friedman, B. Van Essen, C. Ebeling, D. Grossman, and S. Huack. Macah: a "C-level" language for programming kernels on coprocessor accelerators. Technical report. University of Washington, Department of CSE, 2008.