Today: • What are SNARKs?
• Overview of SNARK design landscape.
• If time: cost of key operations.

A SNARK — Argument of Knowledge. (prover cannot find convincing proofs of false statements)

- Lets an untrusted prover prove that it knows a "witness".

succinct
- A bunch of digital signatures authorizing a bunch of transactions.

"rollup"

Proof is short and fast to verify.

- A secret key ("password") that "matches" a public key.

A SNARK is ZK if it reveals no information

about privacy: private payments (Tornado Cash, Aztec)

without the witness
(besides that the witness
satisfies the claimed
property).

Developer writes a "witness checking
procedure" that takes as input
the witness w that the prover
claims to know and verifies the
witness, (e.g. Rust program)

"machine code of SNARKs"

Front-end. spits out an intermediate
representation (IR), circuits or
constraint systems. R1CS, AIR, Plonkish constraints.

IR is fed into a backend

SNARK is analog
of your laptop.
Apple to the "machine
code" circuit or

code ... constraint system)

Example front-ends (zKDSLs)

- Bellman and Circom, Halo2
  - witness checking procedure is specified in a low-level hardware language
- Zokrates, xJsnark, Cairo-VM
- Noir
- Rust

machine code for a simple CPU like RISC-V

backend for RISC-V

RISC0, SP1, Jolt.

Backends

3 step design process

1) Design a polynomial IOP
2) Design a polynomial commitment scheme
3) Apply Fiat-Shamir

One exception: Groth16 very good verifier costs and its predecessors

"linear PCP" + something like KZG commitments

Can (Fiat-Shamir)

PIOP: An interactive protocol where P may send to the verifier V (large) polynomials (n in the billions, trillions)

$q(X) = 1 + X + 3X^2 + 4X^3 + \ldots + X^n$

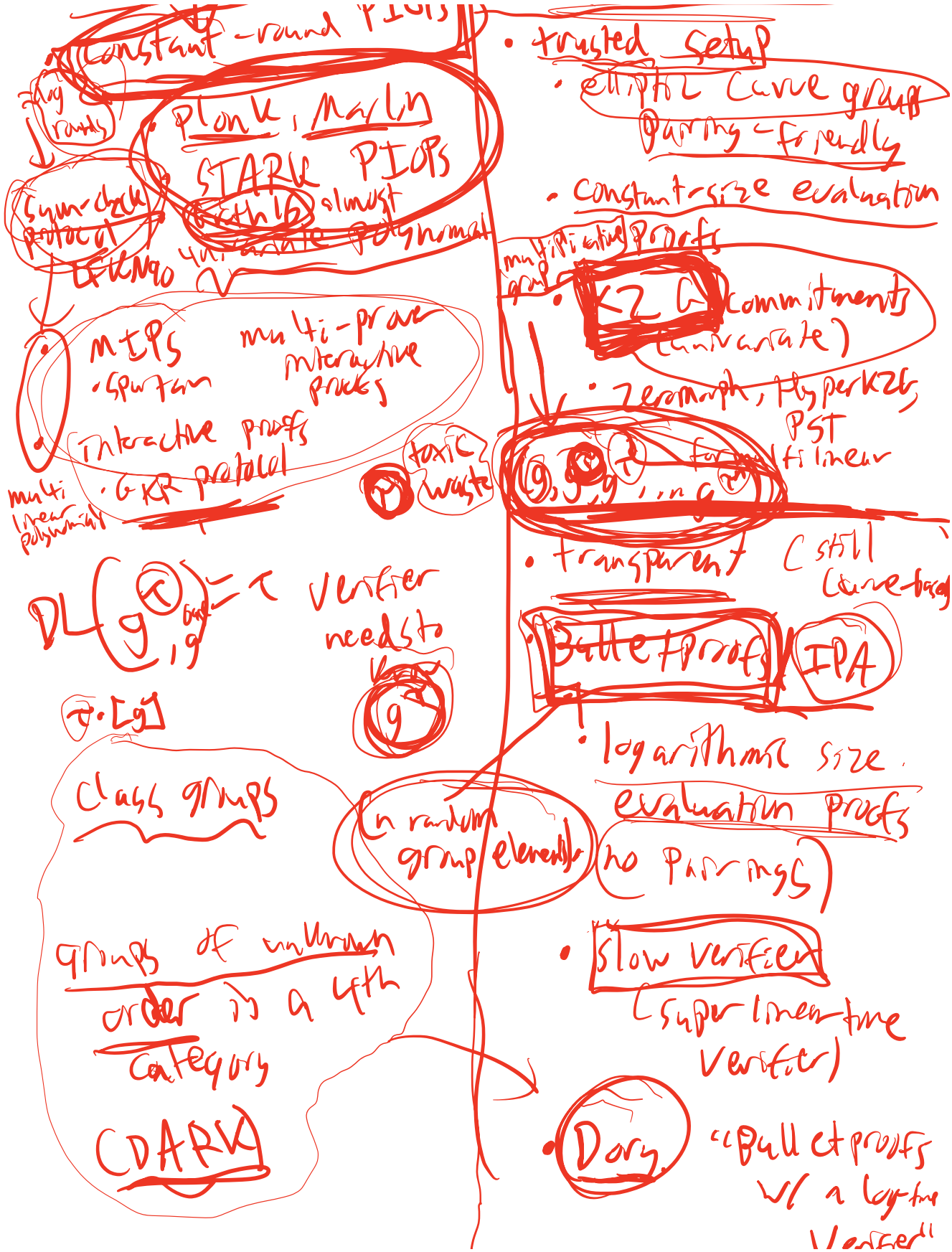
Circuit size n or bigger.

- We give V a query access to q. V is allowed to evaluate q at one point r: $q(r)$ $\neq$ $q'(r)$ if $q \neq q'$

A **polynomial commitment** is what you need to turn a PIOP into a SNARG/SNARK argument.

Example PIOPs

Example polynomial commitments


constant-round
Plonk, Marlin
STARK
PIOPs
almost
sum-check protocol
multivariate polynomial
MIPs
Spartan
multi-prover interactive proofs
interactive proofs
GKR protocol
multi linear polynomial
trusted setup
elliptic curve group
pairing-friendly
constant-size evaluation proofs
multiplicative group
KZG commitments
(univariate)
Zeromorph, HyperKZG, PST
(multilinear)
toxic waste
transparent
(still elliptic curve-based)
Bulletproofs
IPA
logarithmic size evaluation proofs
(no pairings)
slow verifier
(super linear-time verifier)
Dory
"Bulletproofs w/ a log-time verifier"
verifier needs to know
class groups
groups of unknown order
in a 4th category
CDARK
(a random group elements)

Next time:

the main operations that SNARK helper do,

- Bulletproofs
  - proofs are 10x-15x bigger than Bulletproofs.
- Hyrax. square-root size proofs.

For Vertex, v26

the main thing is Pairings

The proof is 3 to 10 group Elements,

V does 3 pairings, each pairing is 1000s of group ops.

- Hashing. based, only universal
- FRI
  - proof size is $\approx O(\lambda \cdot \log^2(n))$, where $\lambda \approx 128$ and n is size of polynomial

Hundreds of KBs.

- Ligero - commit
- Brakedown
- Binius
- FRI-Binius

universal + multilinear

multilinear

Hundreds of KBs

Aztec is using today.
prover runs in a browser
curve-based commitments
clog of space
concretely bigger than FRI.

1) Get the best all worlds for cash using SNARK composition

2) Accumulation / folding schemes

Nova, ProtoStar. Avoid brute force SNARK recursion

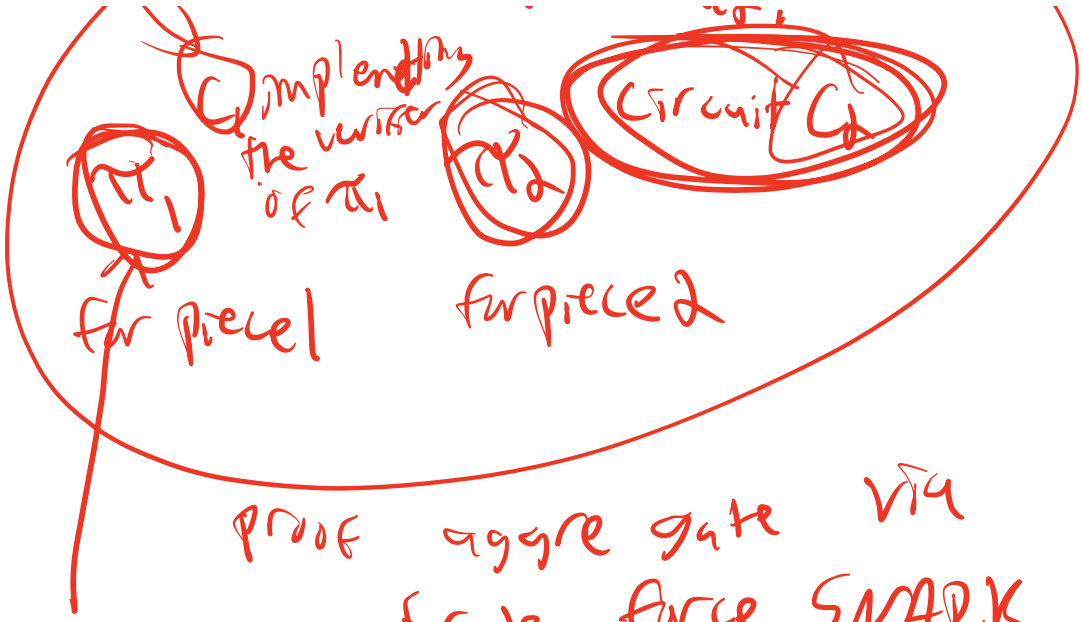
break up big computations into smaller pieces that brute-force SNARK recursion

Useful for controlling prover space.

chop a big computation into small pieces, prove each piece separately, aggregate the proofs, continuations.

the witness for C_1 is π_1.

prove that I know π_1 and V_1



proof aggregation via brute force SNARK recursion

a proof that prover knows a "witness" that it ran the first 2^{20} RISC-V cycles correctly

~