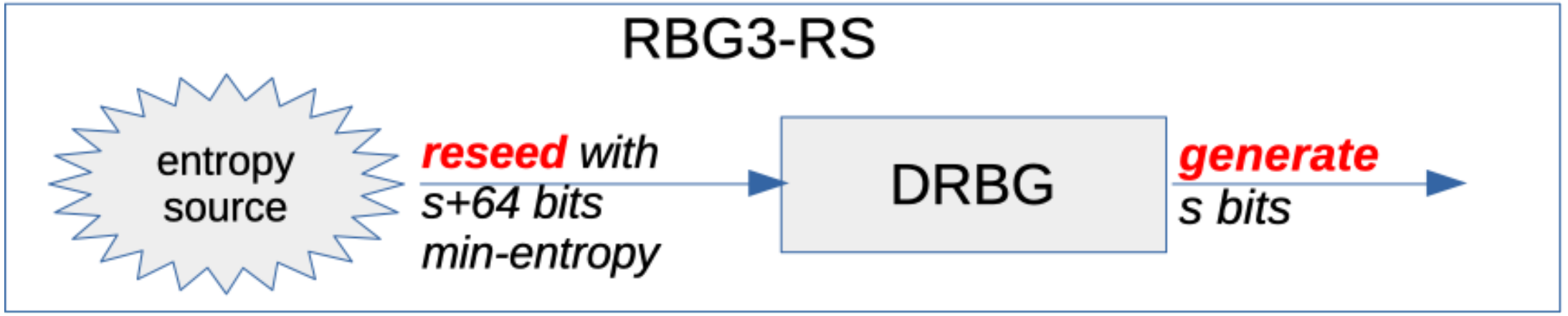
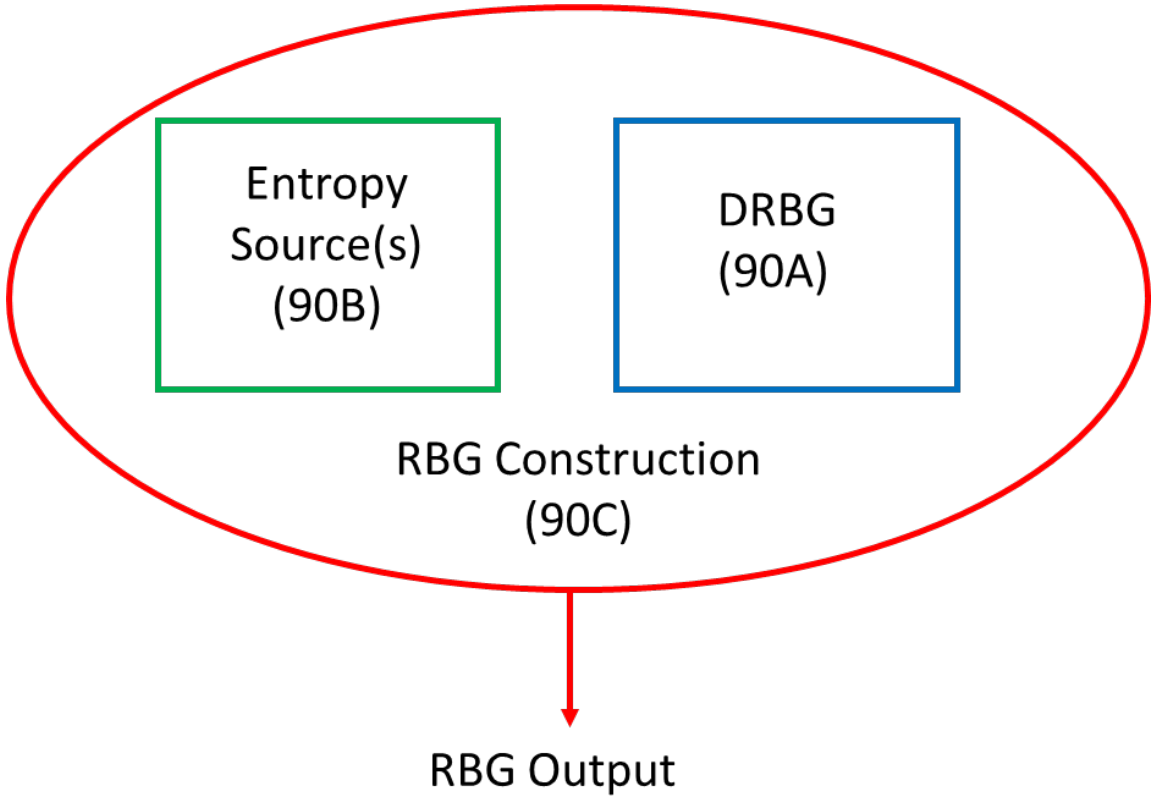
RBG3-RS

John Kelsey, NIST and KU Leuven



SP 800-90

- 90A: DRBGs
  - Deterministic
  - Requires unpredictable seed
- 90B: Entropy sources
  - Nondeterministic
  - Provides bit strings
  - With known amount of entropy
- 90C: Random bit generators
  - Whole construction
  - Provides random bits on demand
  - Different constructions = different security guarantees

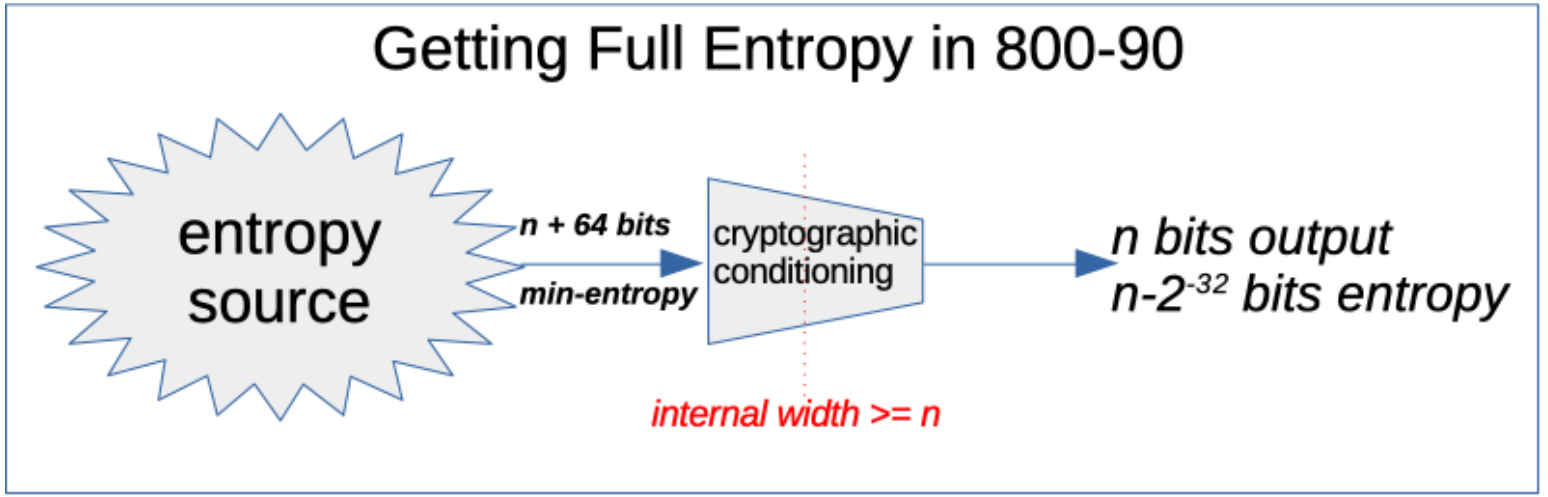


RBG Constructions

- 90C: different ways to build an RBG
  - Different engineering requirements
  - Different security and performance traits

- RBG1 = externally seeded DRBG
- RBG2 = internally seeded DRBG
- RBGC = chained DRBGs
- RBG3 = full-entropy RBG
  - RBG3-RS = RBG2 that reseeds faster than it outputs
  - XOR = RBG2 XORed with full entropy source

Full Entropy???



Discussed yesterday

- Each bit of output has $(1-2^{-32})$ bits min-entropy

- Given 2^{64} output bits, can't distinguish from ideal random
  - Even with unlimited computation

Minimal trust of cryptographic primitives

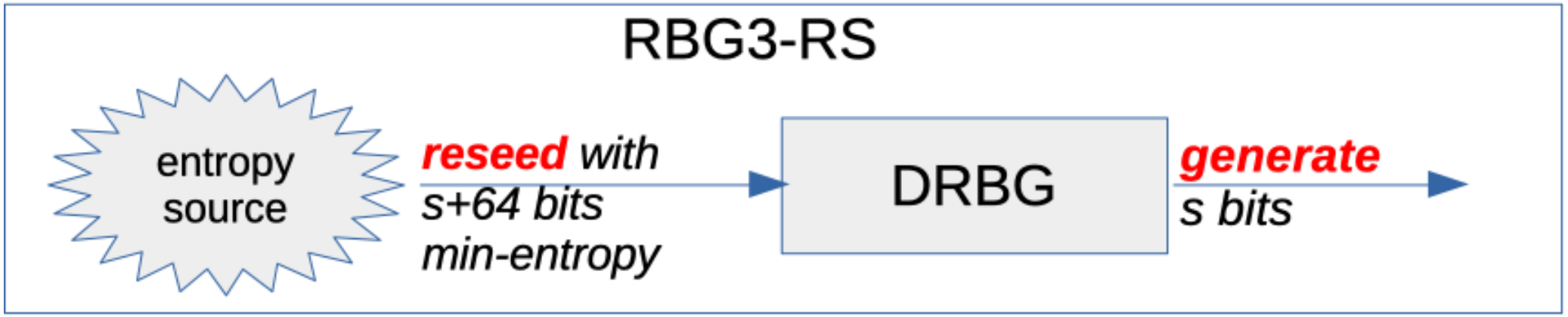
See NIST-IR 8427 for justification and analysis

RBG3-RS

- Provide full entropy by continually reseeding a DRBG
- Requirement for full entropy:

  s +64 bits min-entropy in → s bits out

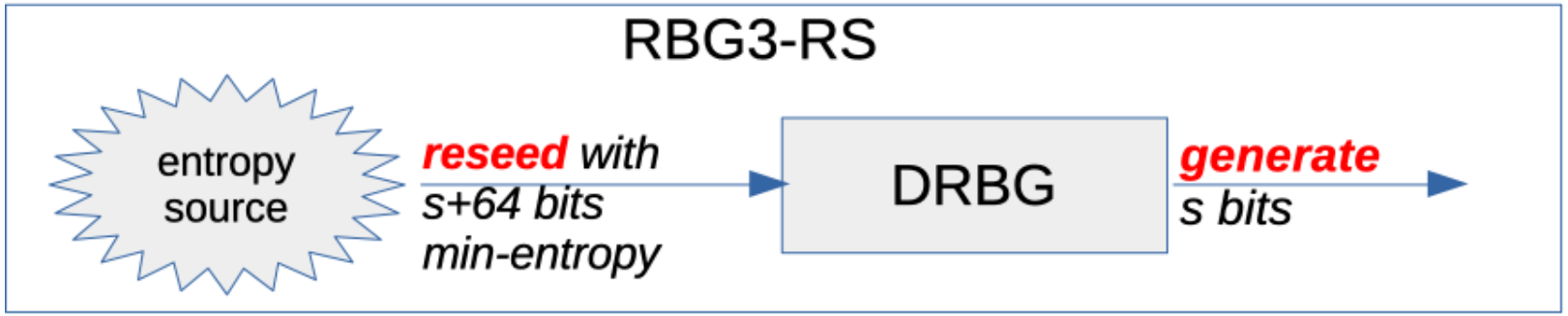
  s = security strength of DRBG



So what's the problem?

We want to output s bits per generate function

… but reseed does not provide enough fresh entropy



- Reseed normally gets s bits min-entropy
  - Allowed to get more but not required
- Can't output more than s bits at a time
  - All DRBG can support

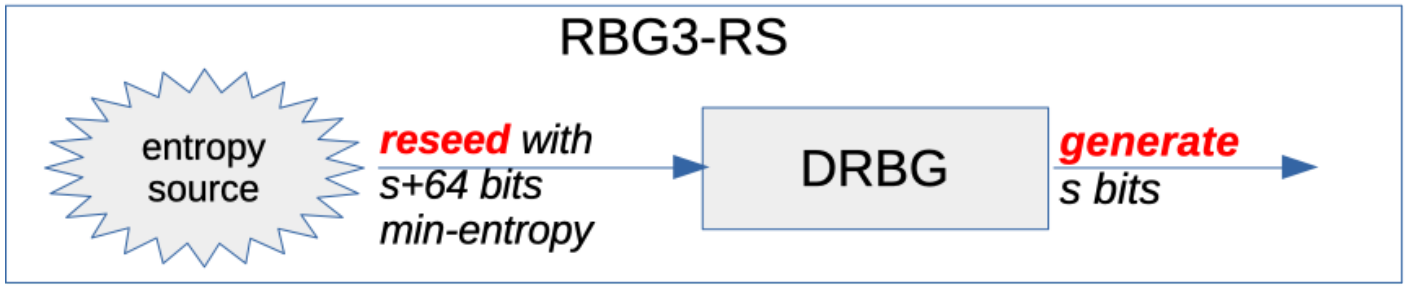
Changes from previous draft 90C

Draft 90C:

- lots of complicated options
- Complex spec
- Validation people found it confusing

New 90C:

- Only two ways to do it
- Much simpler spec



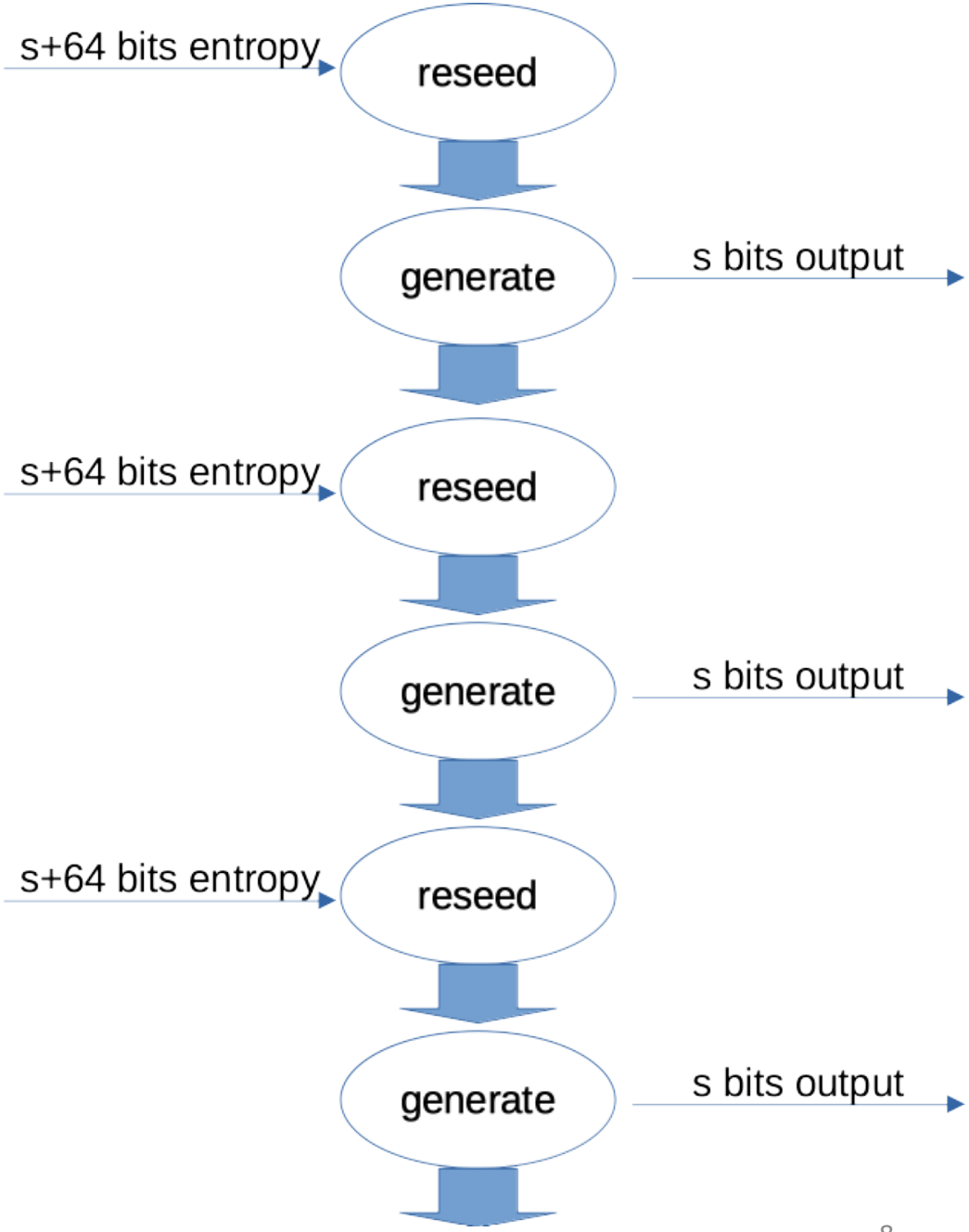
RBG3-RS

DRBG with s bit security strength:

For each s bits required:

- Reseed with s+64 bits entropy
- Generate s bits output

RBG3-RS call generate arbitrary number of output bits in this way!



Problem: Reseed function in 90A

- DRBG reseed function (s bit security):
  - Take optional additional input from caller
  - Draw **at least** s bits entropy from source → entropy input
  - Combine:

    entropy input, additional input, internal state → new internal state

Getting s+64 bits entropy into reseed is the tricky part!

Two ways to do it

1. Change DRBG implementation
   - Reseed draws s+64 bits entropy (instead of just s bits)
   - This was always allowed (no rule against too much entropy)
   - CTR-DRBG without derivation function already guarantees this

2. Put extra 64 bits entropy into additional input to reseed
   - Reseed gets s bits entropy internally, additional input gets 64 bits

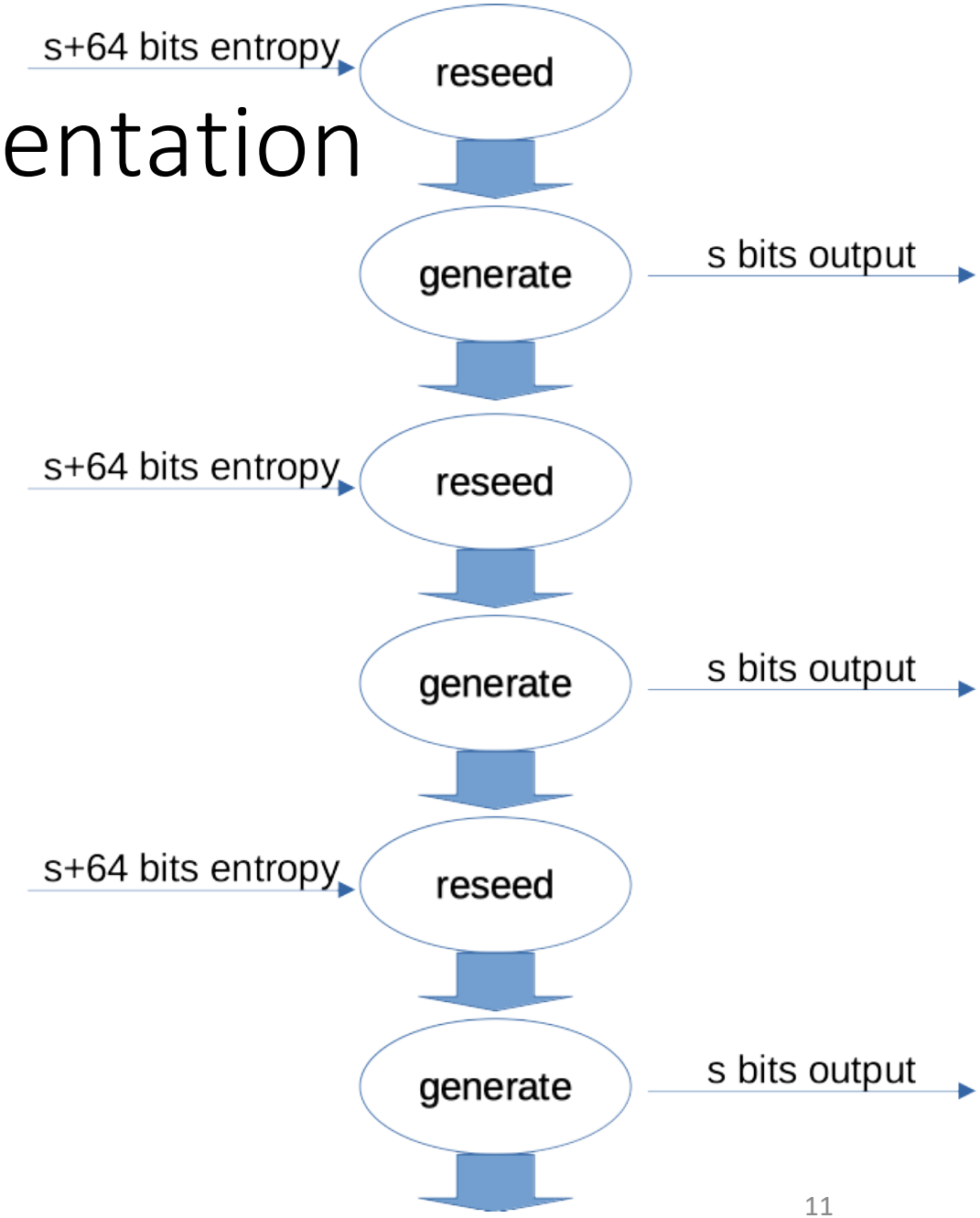
Strategy 1: change implementation

Original:

- Reseed(ai):
  - SEED = Get_entropy (s)
  - STATE = F(STATE, SEED, ai)

New:

- Reseed(ai):
  - SEED = Get_entropy (s +64)
  - STATE = F(STATE, SEED, ai)



This guarantees s+64 bits entropy/reseed

For each s bits required:

- Reseed with s+64 bits entropy ← reseeding with extra entropy
- Generate s bits output

Implementation still compliant with 90A

- It's always permissible to provide more entropy than required!
  - Maybe not obvious this was allowed from 90A
- So we can use same implementation for RBG2

Strategy 2: entropy in additional input

- Suppose we can't change DRBG implementation
  - Stuck with reseed getting only s bits entropy
- We can use additional input in reseed call

For each s bits required:

- ai = Get_entropy (64)
- Call reseed(ai) ← Reseed gets s bits entropy internally + 64 bits entropy from ai
- Generate s bits output

Result: DRBG reseeded with a total of s+64 bits min-entropy

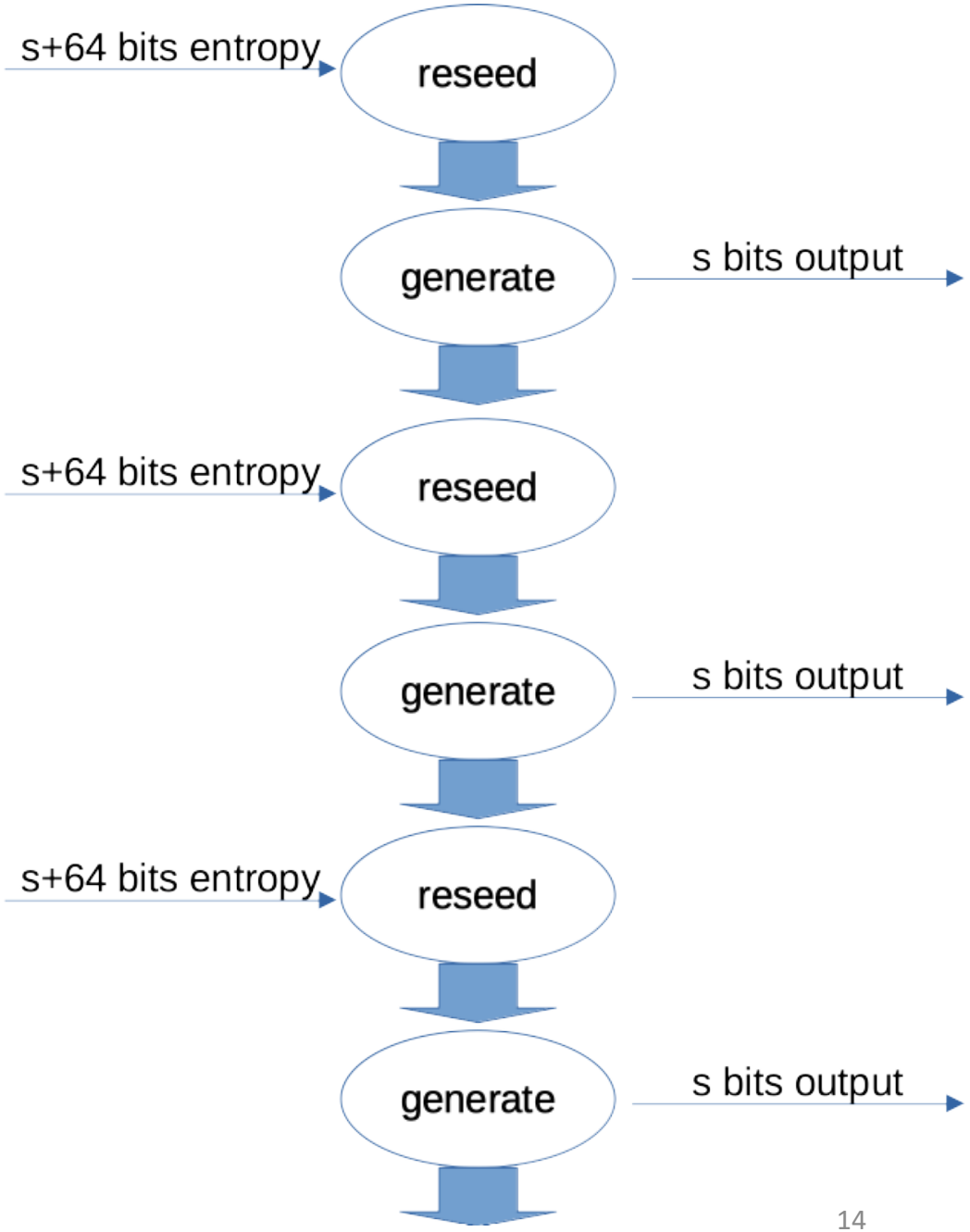
Both techniques work

Strategy 1:

- While more bits needed:
  - Draw extra 64 bits entropy inside reseed
  - Generate s bits output from DRBG

Strategy 2:

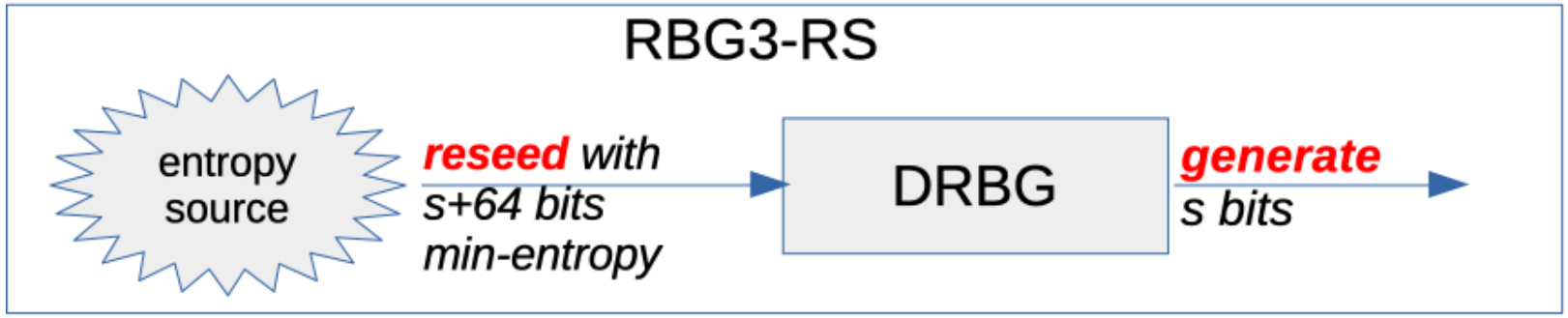
- While more bits needed:
  - Draw extra 64 bits entropy outside reseed
  - Pass in to reseed call as additional input
  - Generate s bits output from DRBG



Simpler spec, fewer options

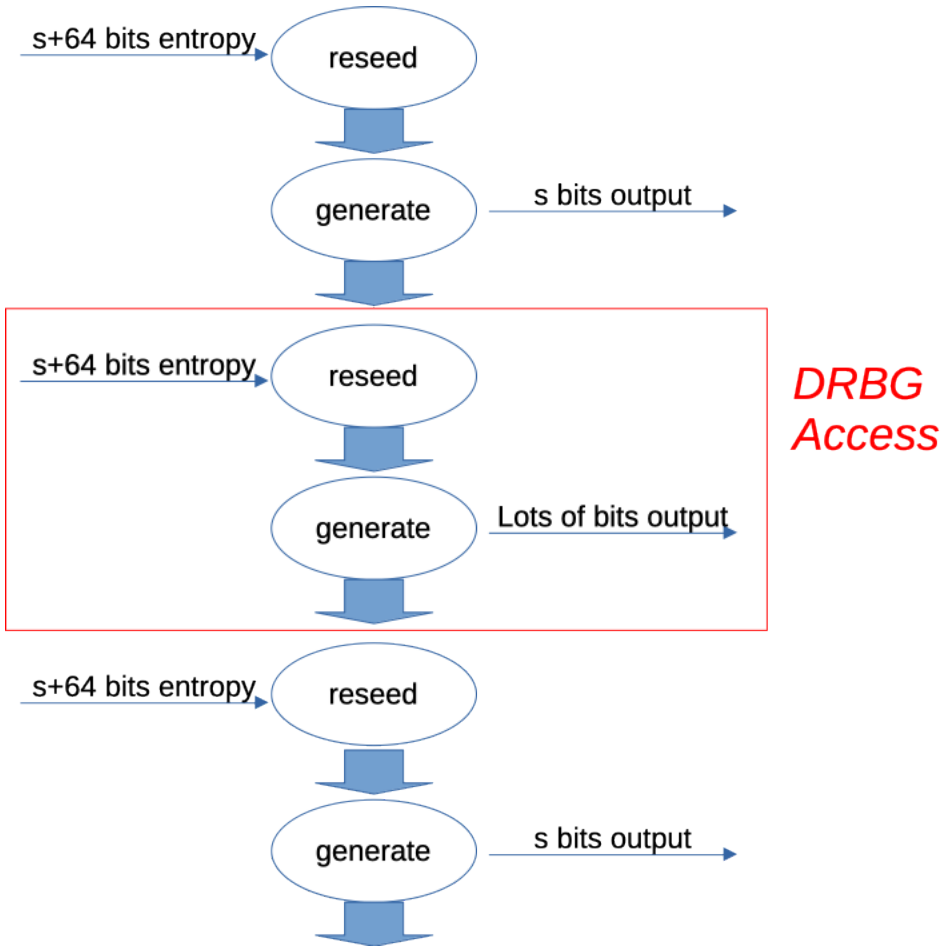
For each S bit output:

- Reseed DRBG with s+64 bits entropy
  - Using either strategy
- Generate s bits output from DRBG



Accessing DRBG

- Full entropy bits slow
  - Outlet rate limited by entropy source
- Sometimes we just want DRBG outputs
  - RBG2 security, not full entropy
- We can do this with RBG3-RS
  - Reseed
  - Generate as many RBG2 bits as you need



Reseed needed to guarantee full entropy of previous s-bit output

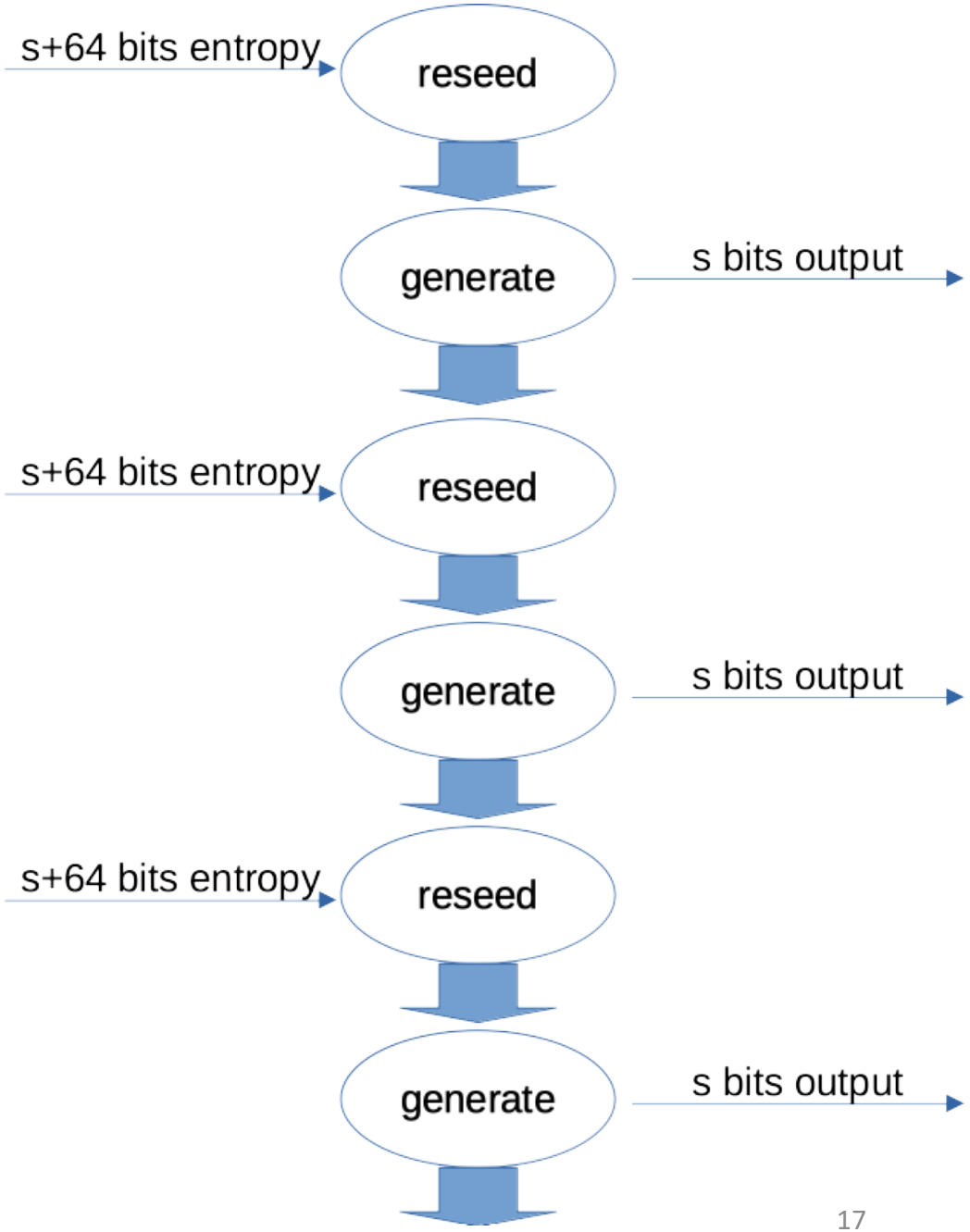
Wrap up

Previous draft:

- RBG3-RS spec too complicated
- Too many options
- Confusing

Now:

- Two options
- RBG3-RS always works the same way



Extras

Why can’t we just do it in Generate call?

Hash_DRBG_Generate Process:

If (*additional_input* = *Null*), then do

$w = \textbf{Hash}(0x02 \,||\, V \,||\, additional\_input)$ ← w is only n bits wide!

Can’t put n+64 bits entropy in!

$V = (V + w) \bmod 2^{seedlen}$

(*returned_bits*) = **Hashgen** (*requested_number_of_bits*, *V*)

$H = \textbf{Hash}(0x03 \,||\, V)$

$V = (V + H + C + reseed\_counter) \bmod 2^{seedlen}$