Low stress version control

“Low stress”?

Imagine a perfect system...
what's so great about it?

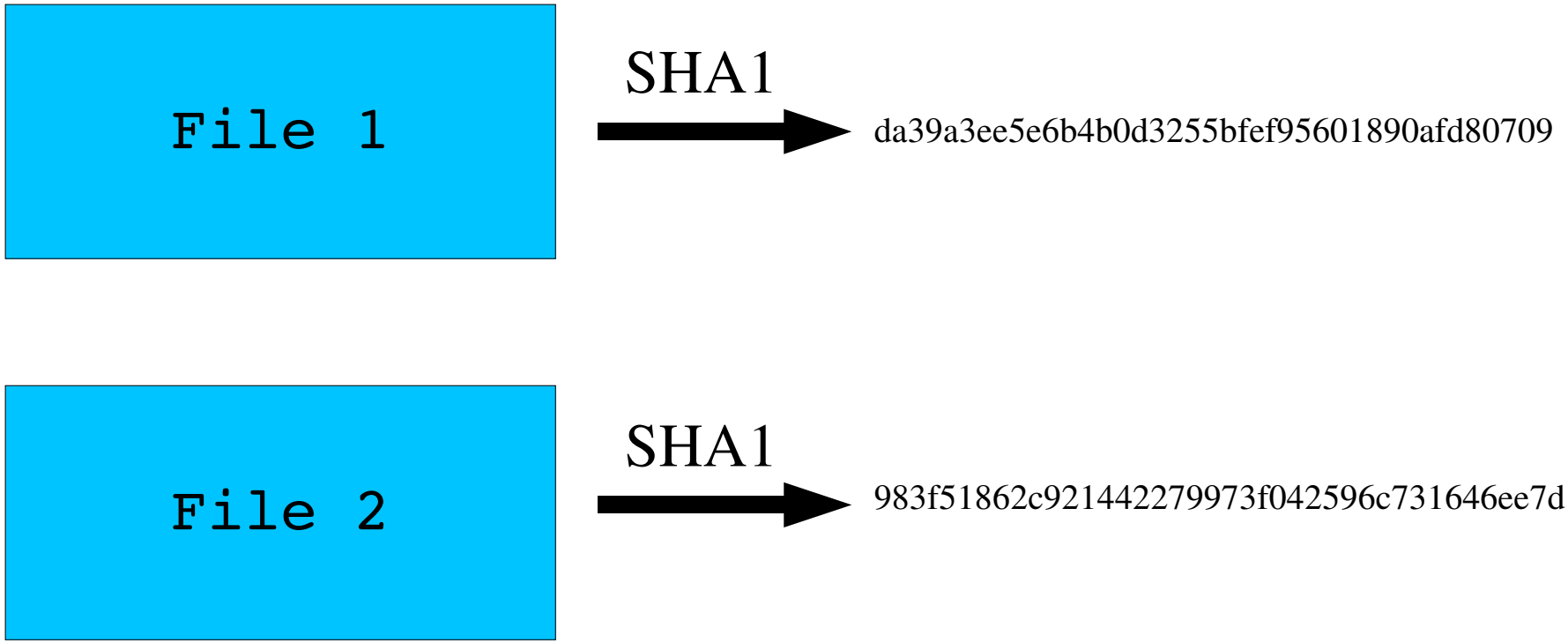
And how does monotone
measure up?

I remain stress-free because...

I can understand it

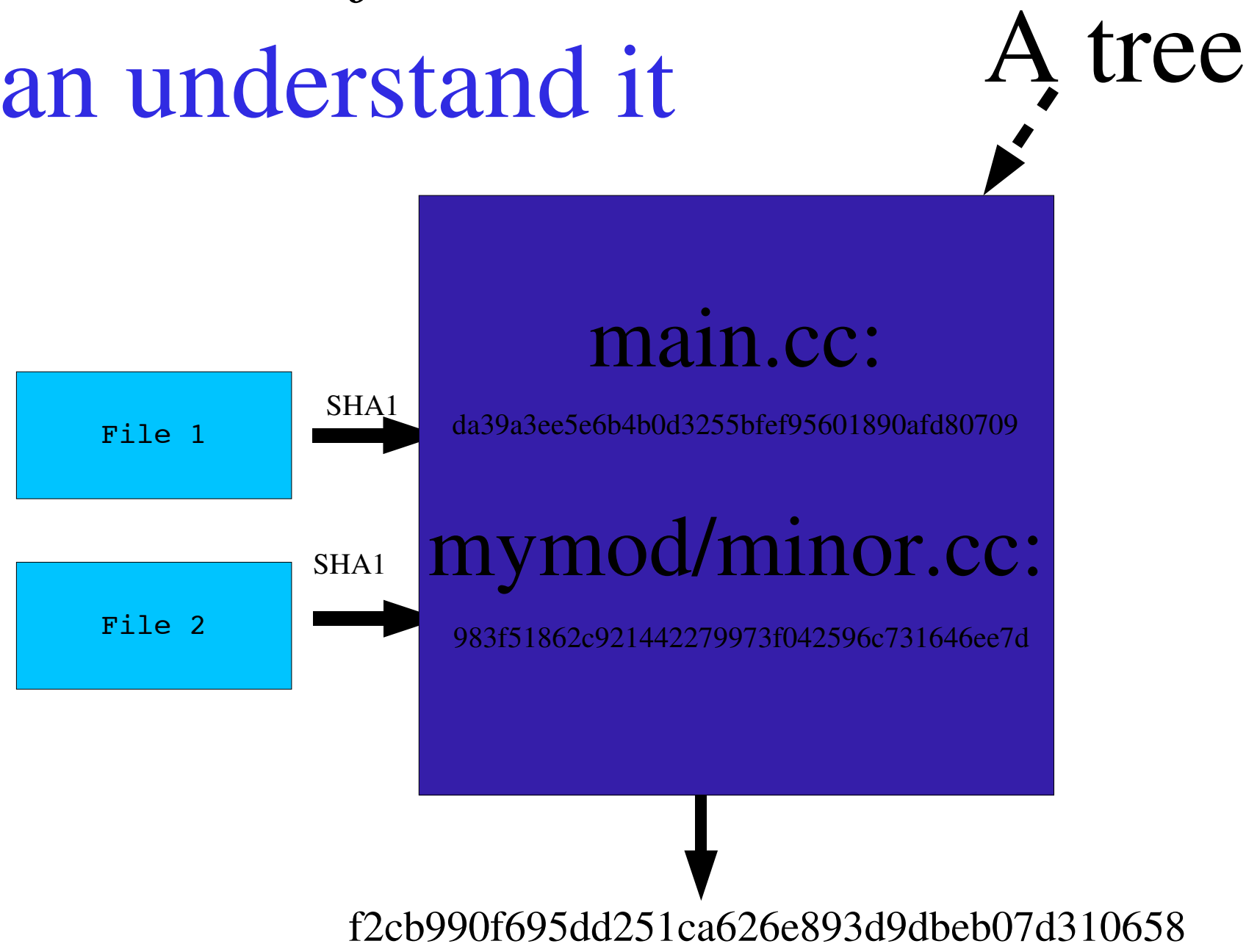
I remain stress-free because...

I can understand it



I remain stress-free because...

I can understand it



I remain stress-free because...

I can understand it

```
format_version "1"

dir ""

   file "main.cc"
content [da39a3ee5e6b4b0d3255bfef95601890afd80709]

dir “mymod”

   file "mymod/minor.cc"
content [983f51862c92144279973f042596c731646ee7d]
```

I remain stress-free because...

I can understand it

But what about *versions*?

I remain stress-free because...

I can understand it

Revision:

```
format_version "1"

new_manifest [65cd5f8b1cd1e10e210d9d966ae3bd1696200295]

old_revision [64fd1c6eb06f48d574ff7433178ba2b9b9138198]


patch "main.cc"
 from [983f51862c921442279973f042596c731646ee7d]
   to [da39a3ee5e6b4b0d3255bfef95601890afd80709]
```

I remain stress-free because...

I can understand it

Revision:

```
format_version "1"

new_manifest [65cd5f8b1cd1e10e210d9d966ae3bd1696200295]

old_revision [64fd1c6eb06f48d574ff7433178ba2b9b9138198]


patch "main.cc"
 from [983f51862c921442279973f042596c731646ee7d]
   to [da39a3ee5e6b4b0d3255bfef95601890afd80709]
```

I remain stress-free because...

I can understand it

Revision:

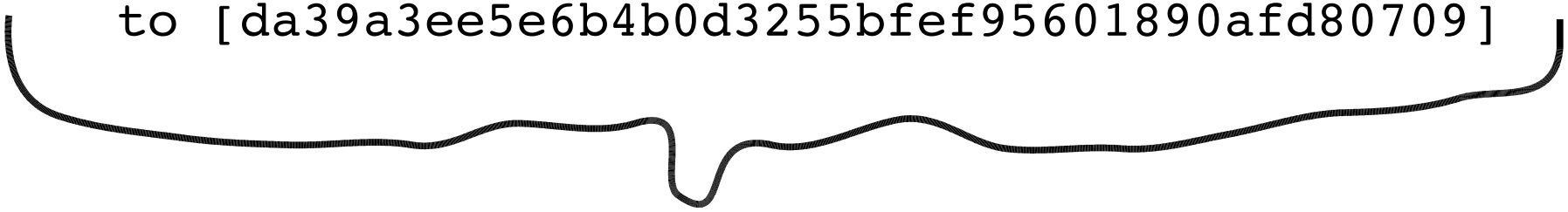
```
format_version "1"

new_manifest [65cd5f8b1cd1e10e210d9d966ae3bd1696200295]

old_revision [64fd1c6eb06f48d574ff7433178ba2b9b9138198]

patch "main.cc"
 from [983f51862c921442279973f042596c731646ee7d]
   to [da39a3ee5e6b4b0d3255bfef95601890afd80709]
```

I remain stress-free because...

I can understand it

Revision:

```
format_version "1"

new_manifest [65cd5f8b1cd1e10e210d9d966ae3bd1696200295]

old_revision [64fd1c6eb06f48d574ff7433178ba2b9b9138198]


patch "main.cc"
 from [983f51862c921442279973f042596c731646ee7d]
   to [da39a3ee5e6b4b0d3255bfef95601890afd80709]
```



983f51862c921442279973f042596c731646ee7d

I remain stress-free because...

I can understand it

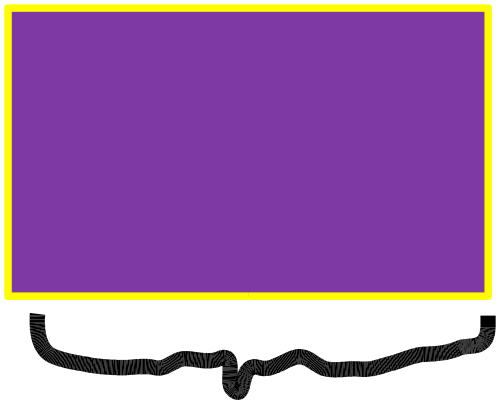
Revision:

```
format_version "1"

new_manifest [65cd5f8b1cd1e10e210d9d966ae3bd1696200295]

old_revision [64fd1c6eb06f48d574ff7433178ba2b9b9138198]

patch "main.cc"
 from [983f51862c921442279973f042596c731646ee7d]
   to [da39a3ee5e6b4b0d3255bfef95601890afd80709]
```

983f51862c921442279973f042596c731646ee7d

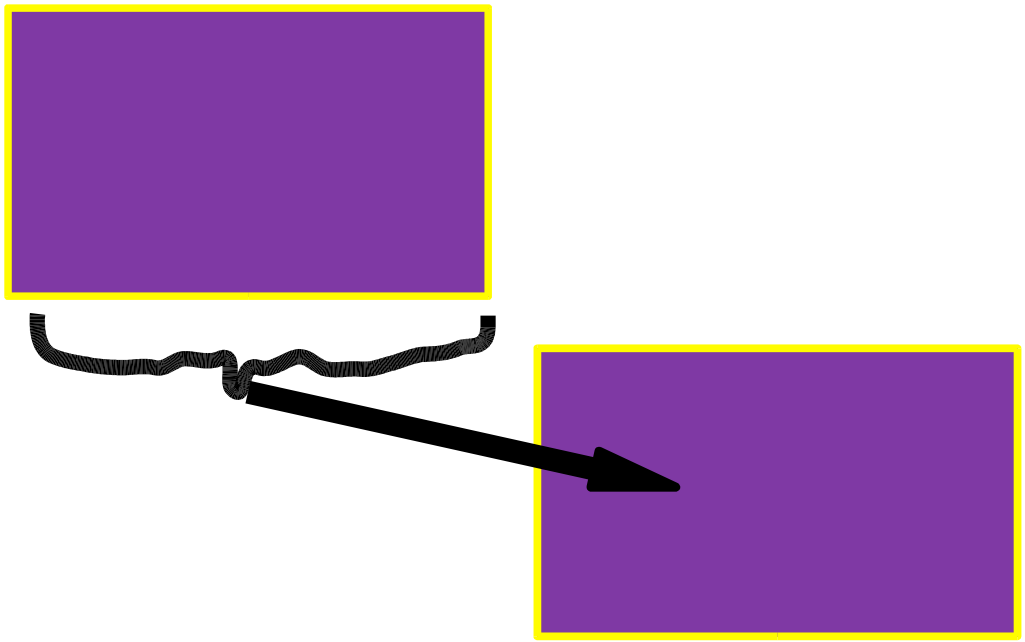
I remain stress-free because...

I can understand it



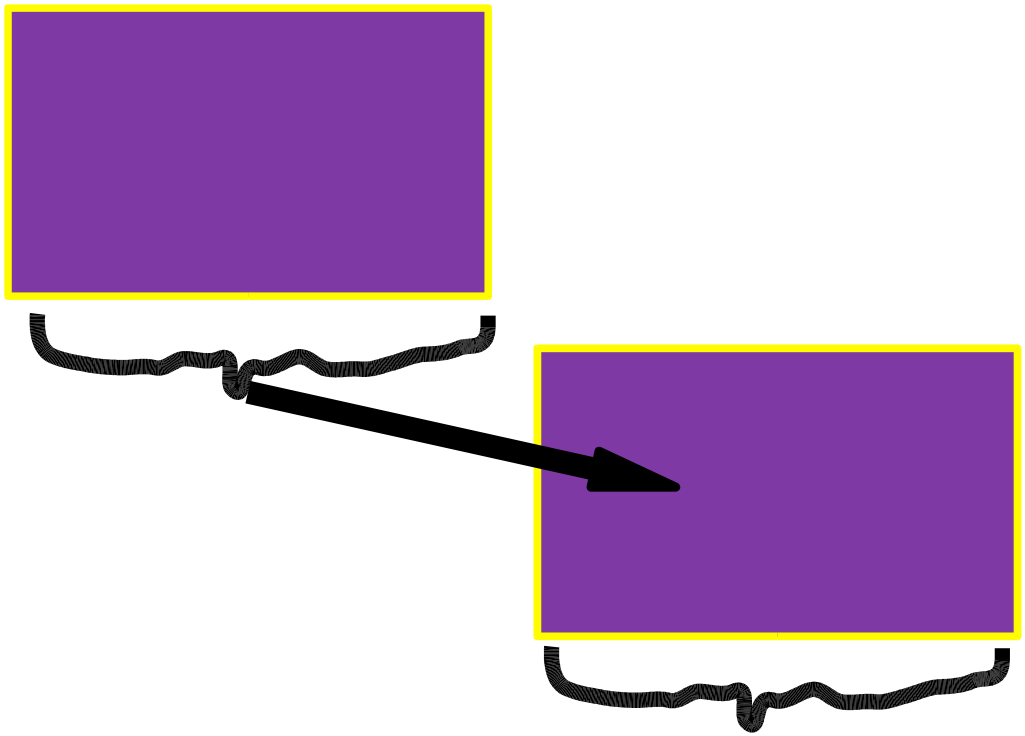
I remain stress-free because...

I can understand it



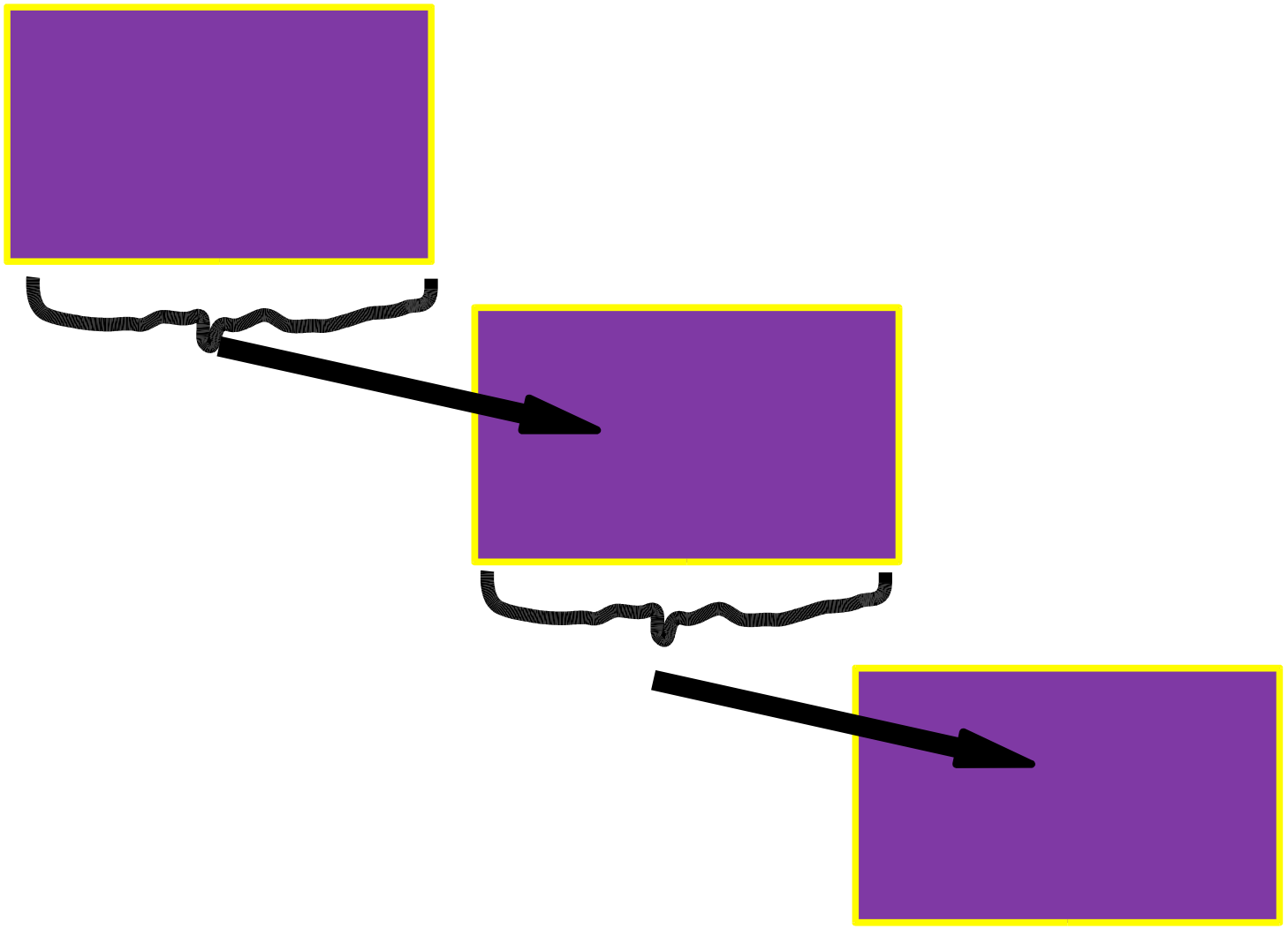
I remain stress-free because...

I can understand it



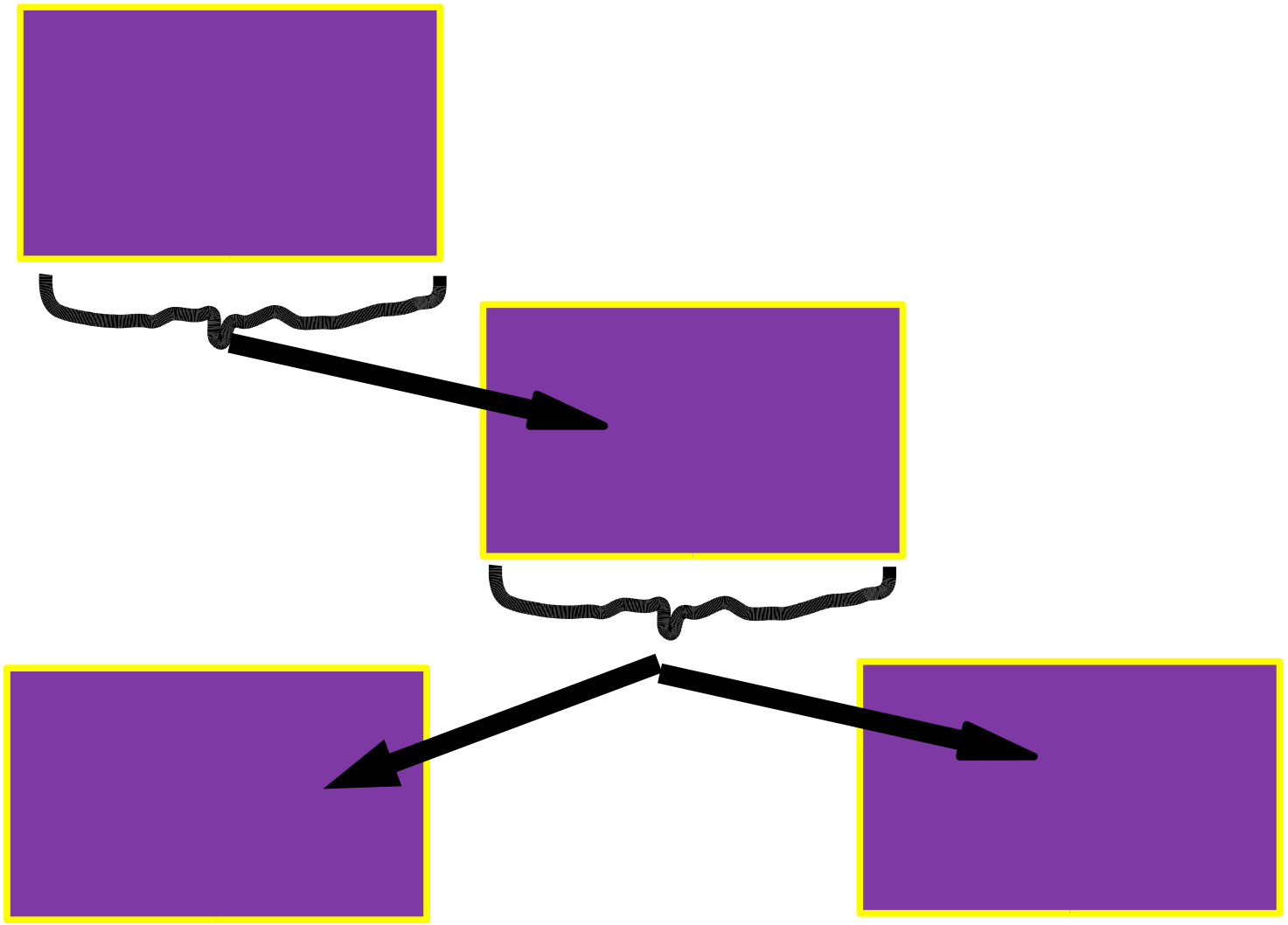
I remain stress-free because...

I can understand it



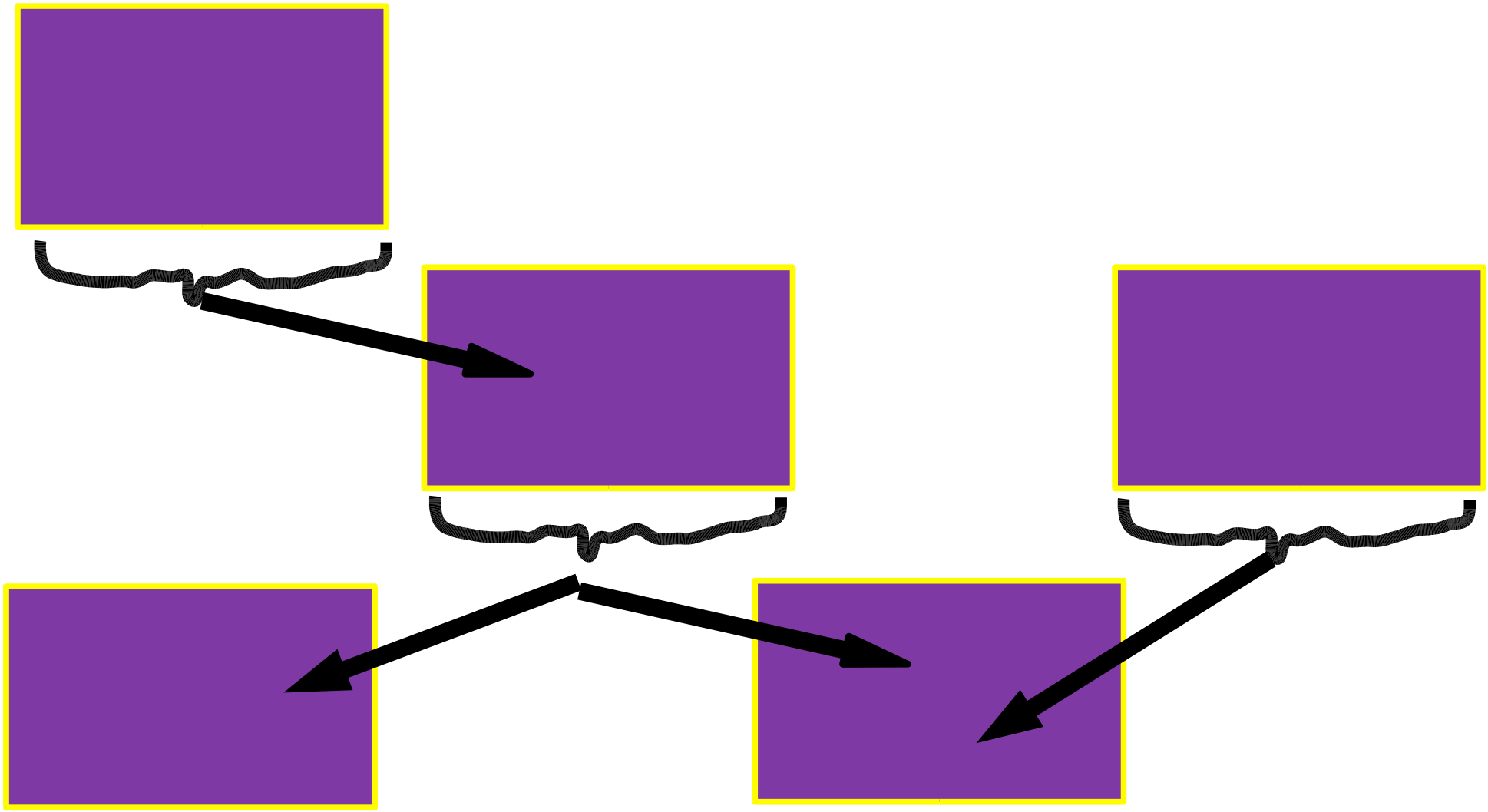
I remain stress-free because...

I can understand it



I remain stress-free because...

I can understand it



I remain stress-free because...

I can understand it

Review:

- files make trees (“manifests”)
- revisions make a graph (DAG) of trees
- every revision has a unique name

I remain stress-free because...

I can understand it

So that's versions... what about *control*?

I remain stress-free because...

I can understand it

Each revision has a cryptographically strong name

--> give each user an RSA key

--> let them attach signed, key/value pairs to each revision

I remain stress-free because...

I can understand it

Each revision has a cryptographically strong name

--> give each user an RSA key

--> let them attach signed, key/value pairs to each revision

= *certs*

I remain stress-free because...

I can understand it

Most important certs:

-- changelog messages

-- branch certs

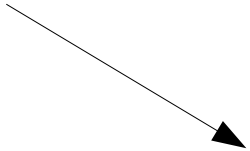
I remain stress-free because...

I can understand it

Most important certs:

-- changelog messages

-- branch certs



A revision is in branch Foo if there is a branch=Foo cert on it

I remain stress-free because...

I can understand it

That's all!

I remain stress-free because...

I can understand it

That's all!

Files, manifests, revisions, certs

(mostly you can even forget about files and manifests)

I remain stress-free because...

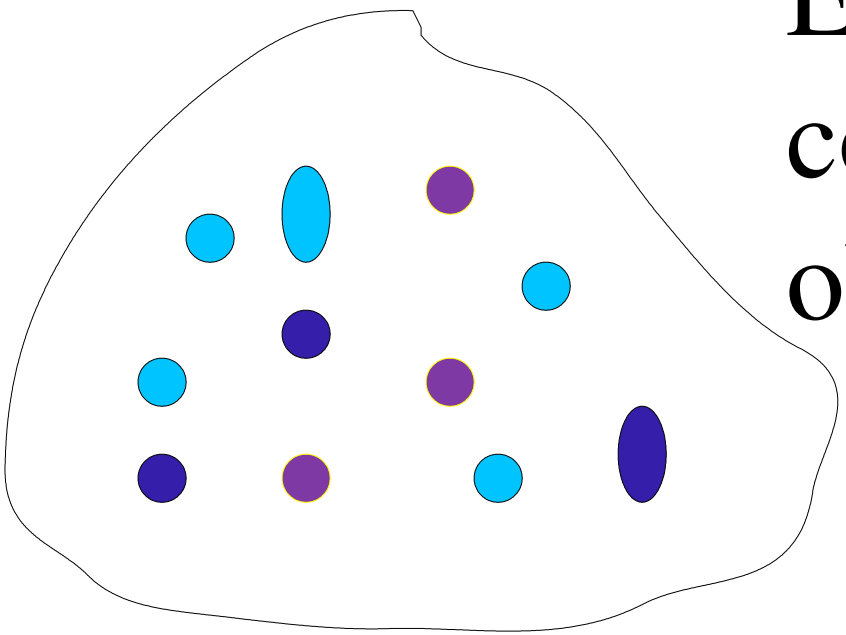
I can understand it

...but where are they?

I remain stress-free because...

I can understand it

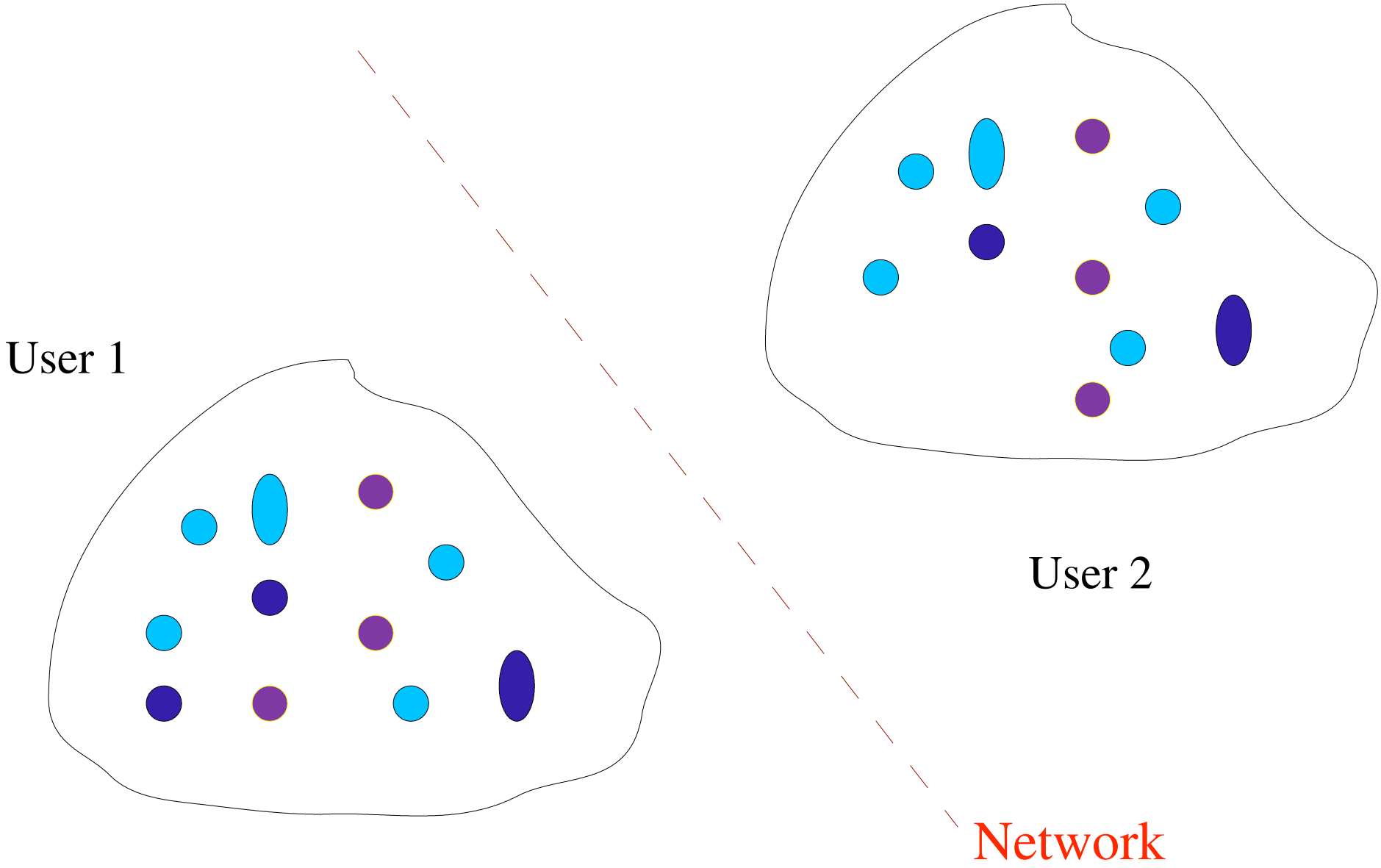
...but where are they?



Each user has a file containing a bag of these objects – the complete history of the project, since they last pulled

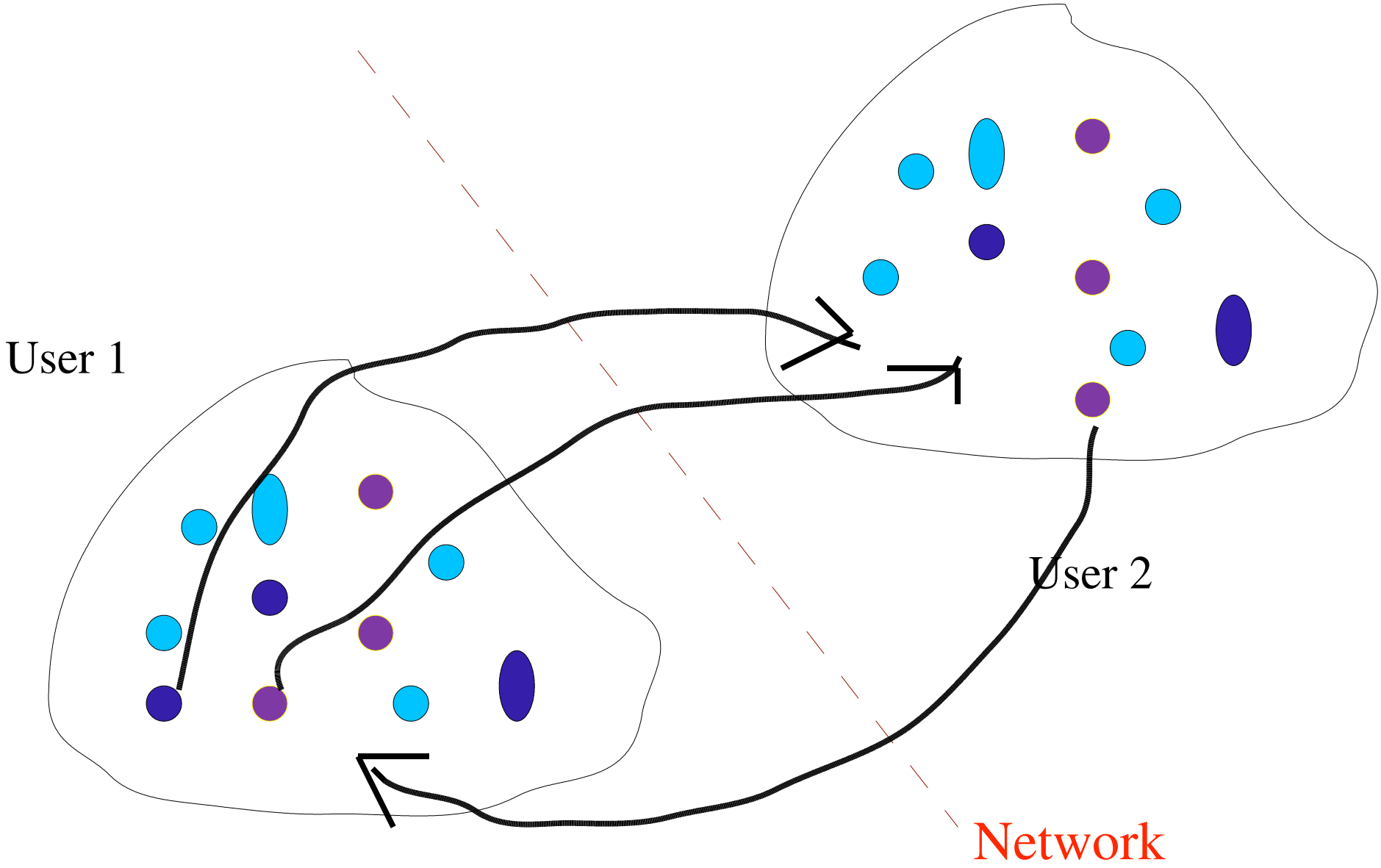
I remain stress-free because...

I can understand it



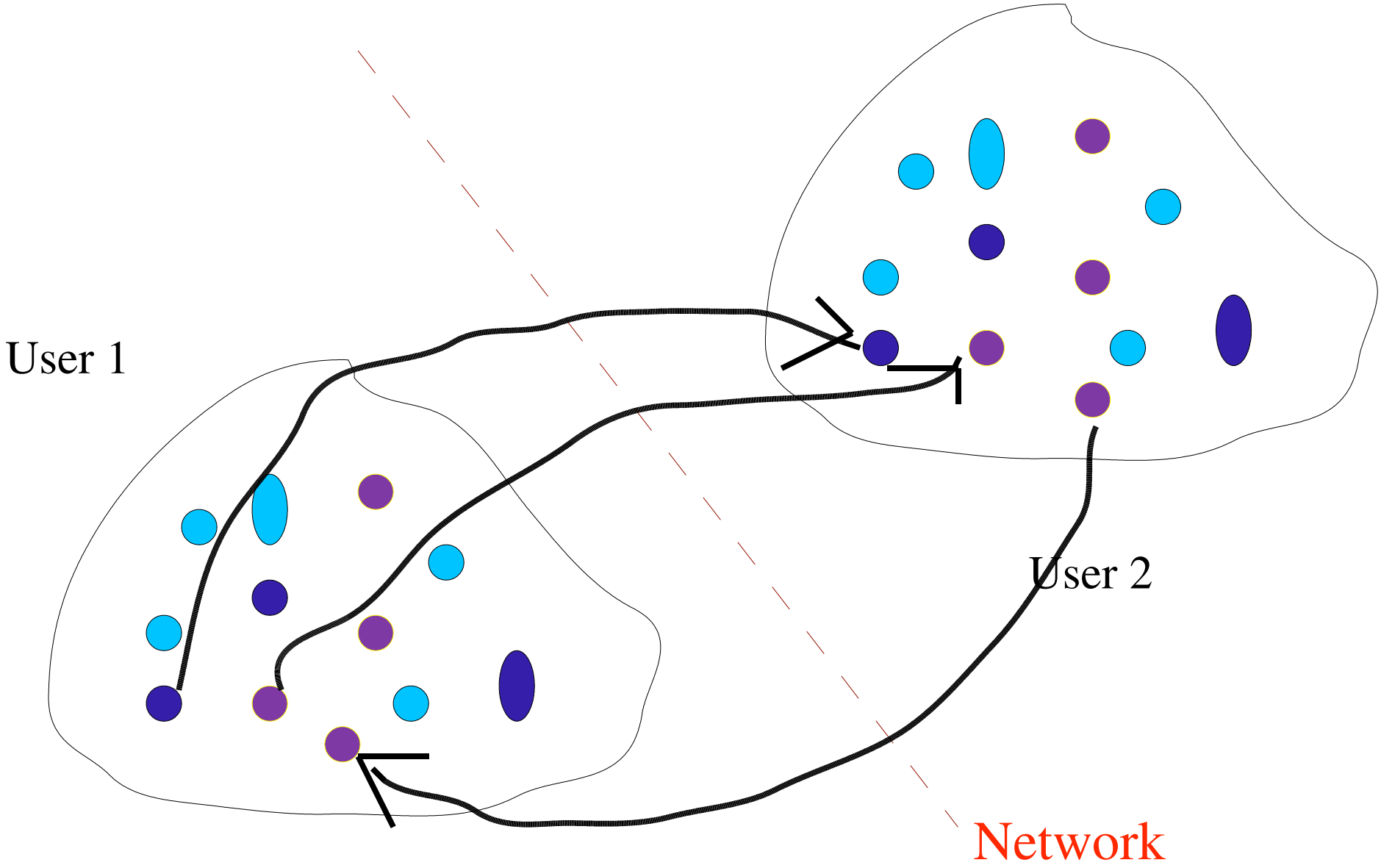
I remain stress-free because...

I can understand it



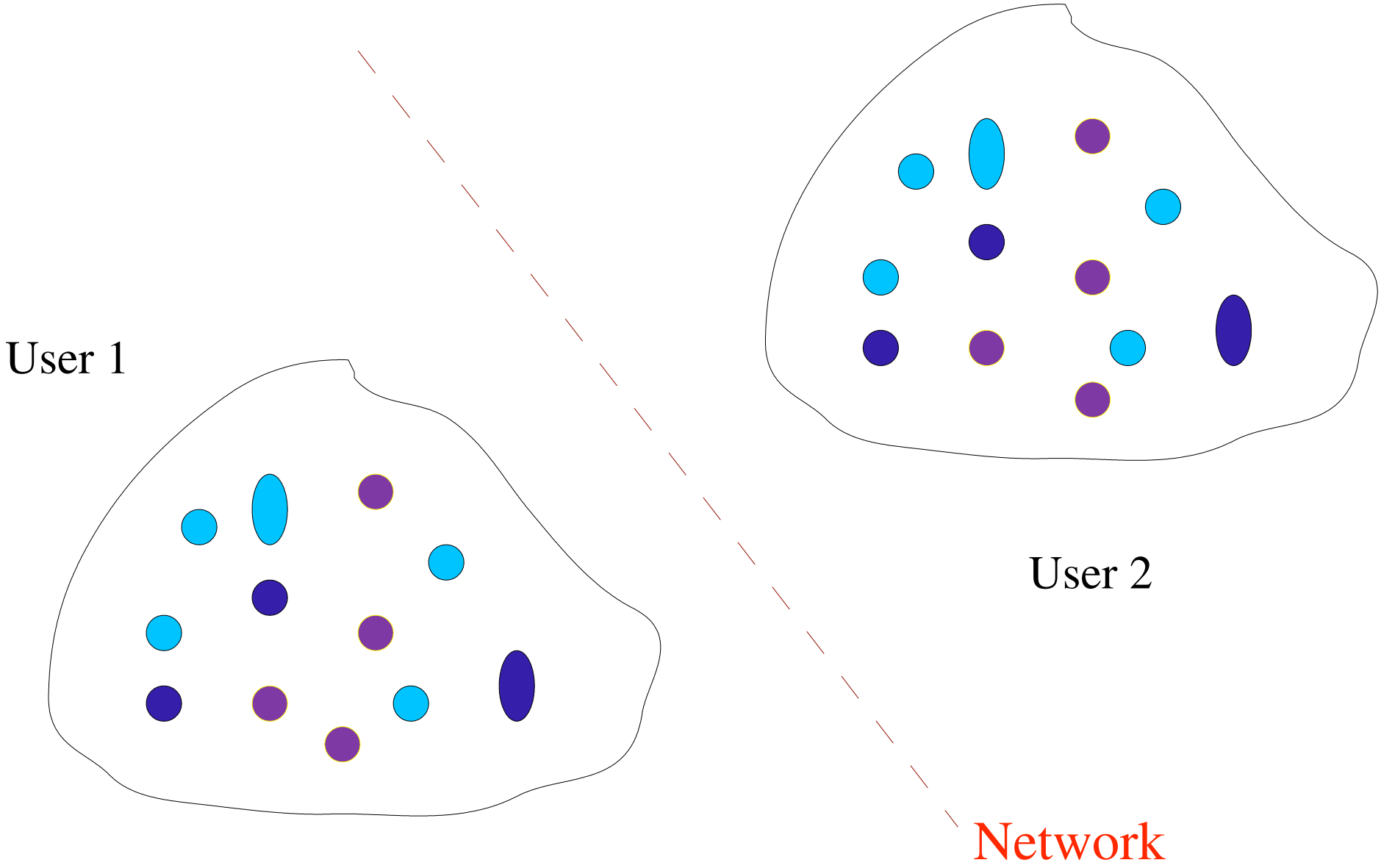
I remain stress-free because...

I can understand it



I remain stress-free because...

I can understand it



I remain stress-free because...

I can understand it

For convenience, “User 2” is almost always a single server (or round-robin cluster) shared by one community.

I remain stress-free because...

I can understand it

But...!

I remain stress-free because...

I can understand it

But...!

The whole history on every developer's hard drive? That's totally unreasonable!

I remain stress-free because...

I can understand it

But...!

You have to hash *everything*, and do public key operations just to find out what branches a revision is in?
That's way too slow!

I remain stress-free because...

I can understand it

But...!

As the repos get big, it will take you forever to find the pieces each side is missing!

I remain stress-free because...

I can understand it

But...!

As the repos get big, it will take you forever to find the pieces each side is missing!

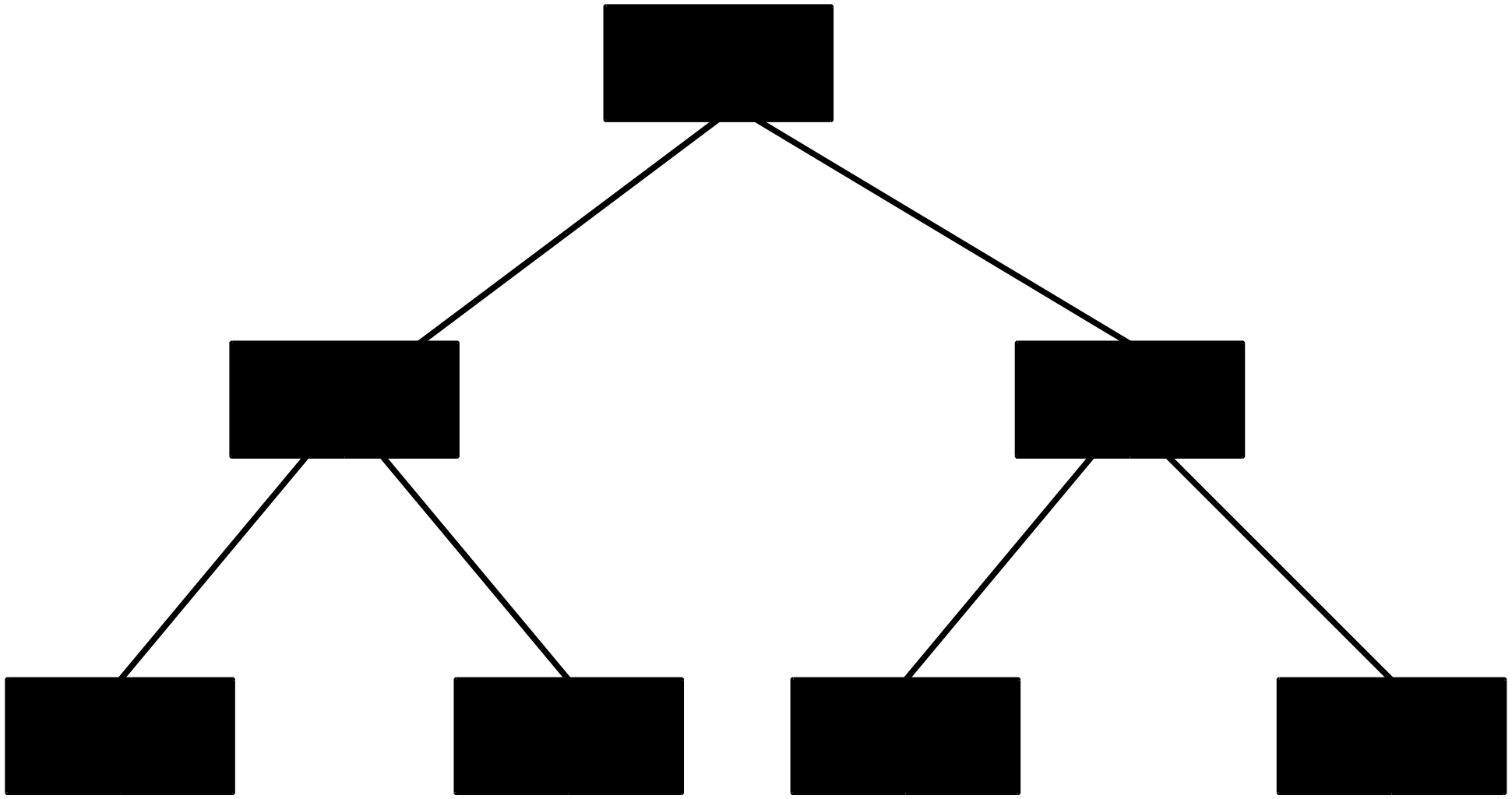
I'm glad you asked...

Digression: merkle tries

001
010
100
110

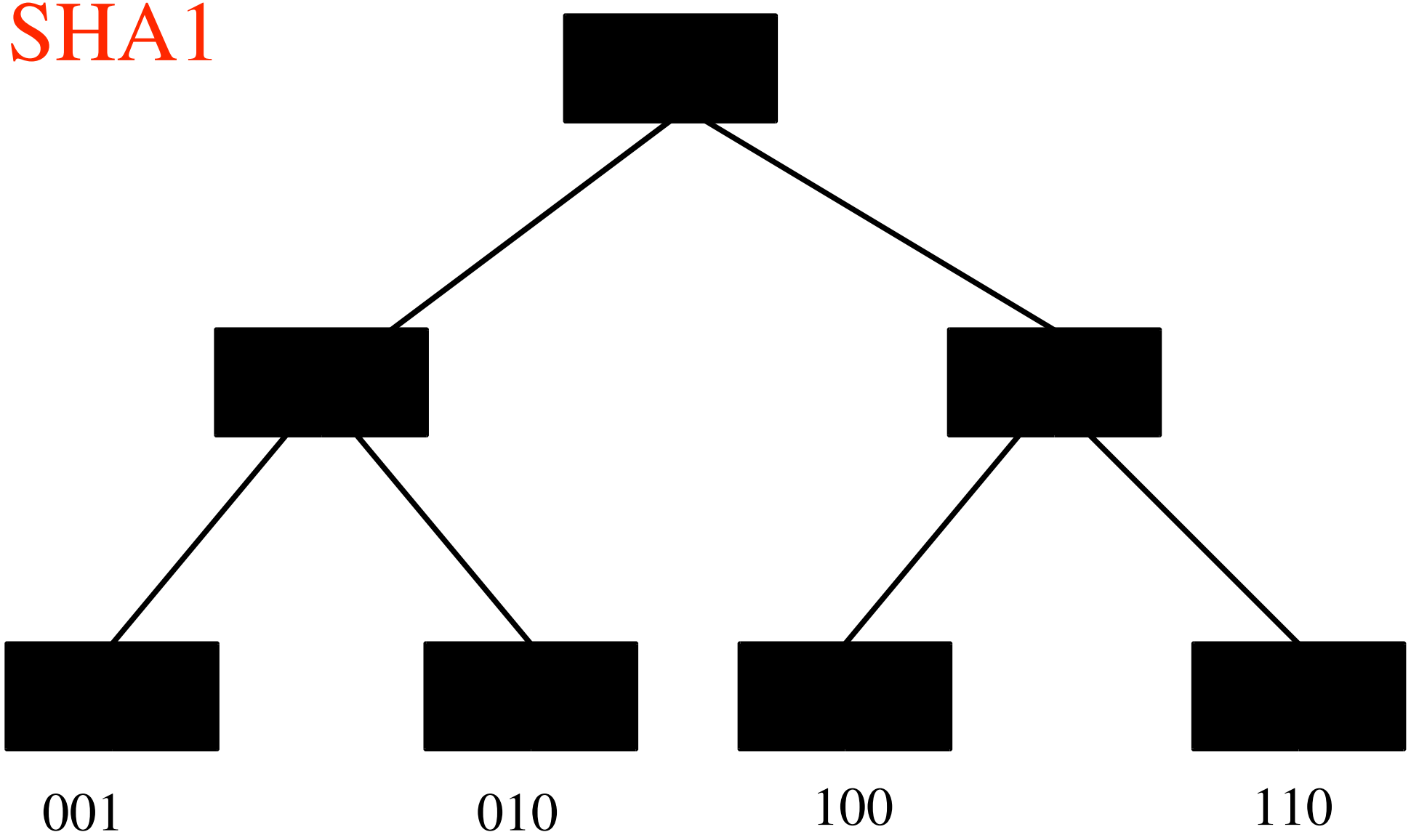
Digression: merkle tries

Digression: merkle tries



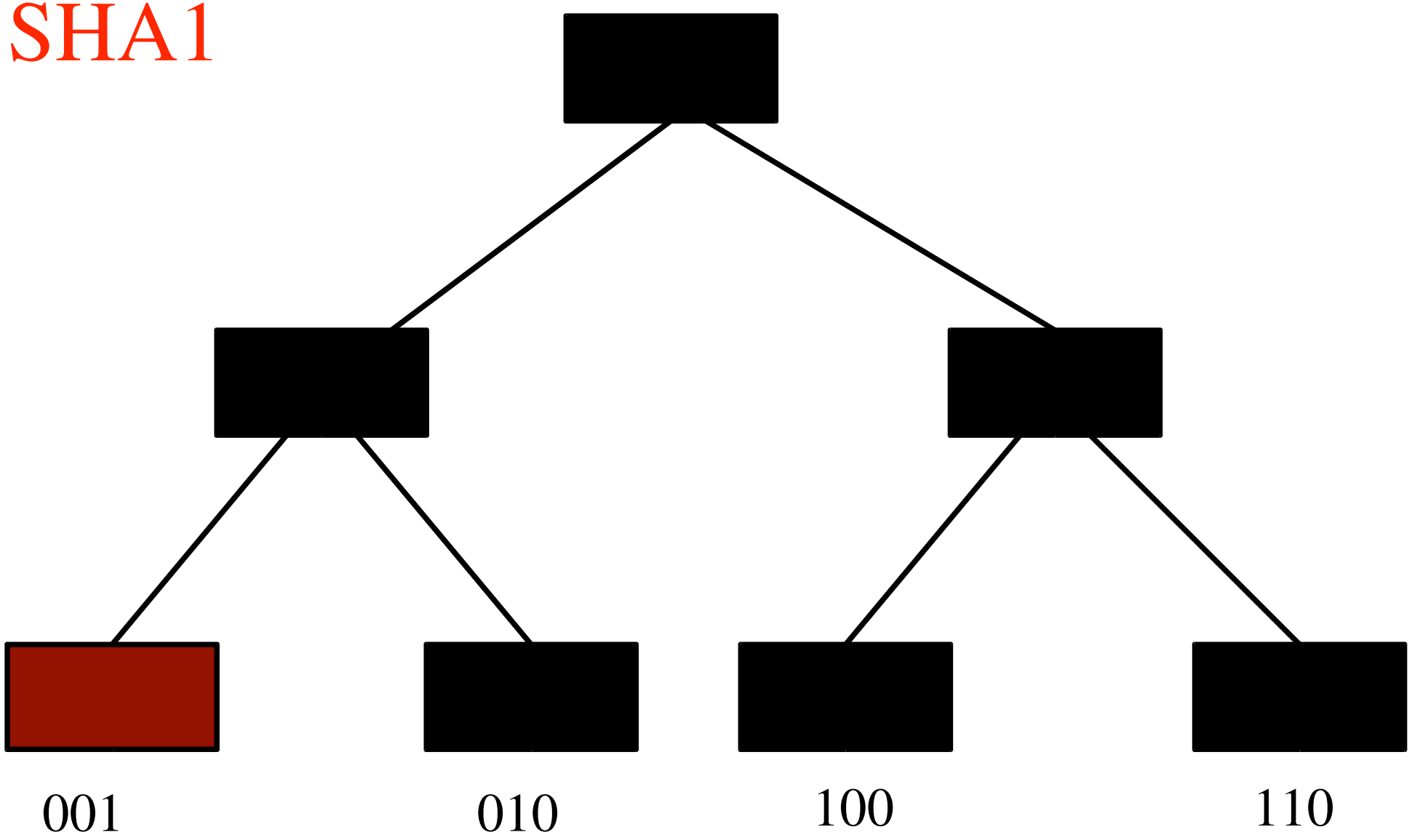
Digression: merkle tries

SHA1



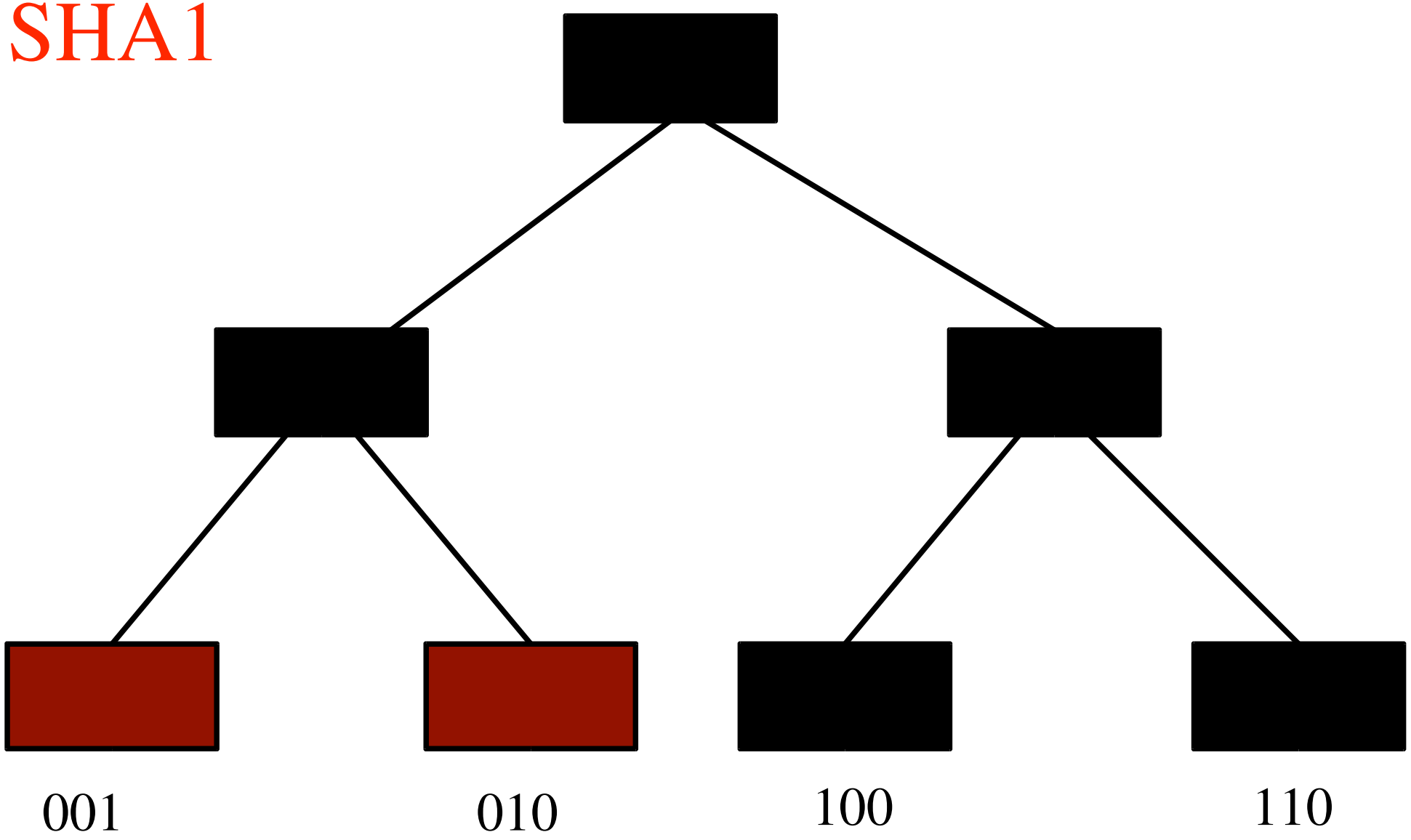
Digression: merkle tries

SHA1



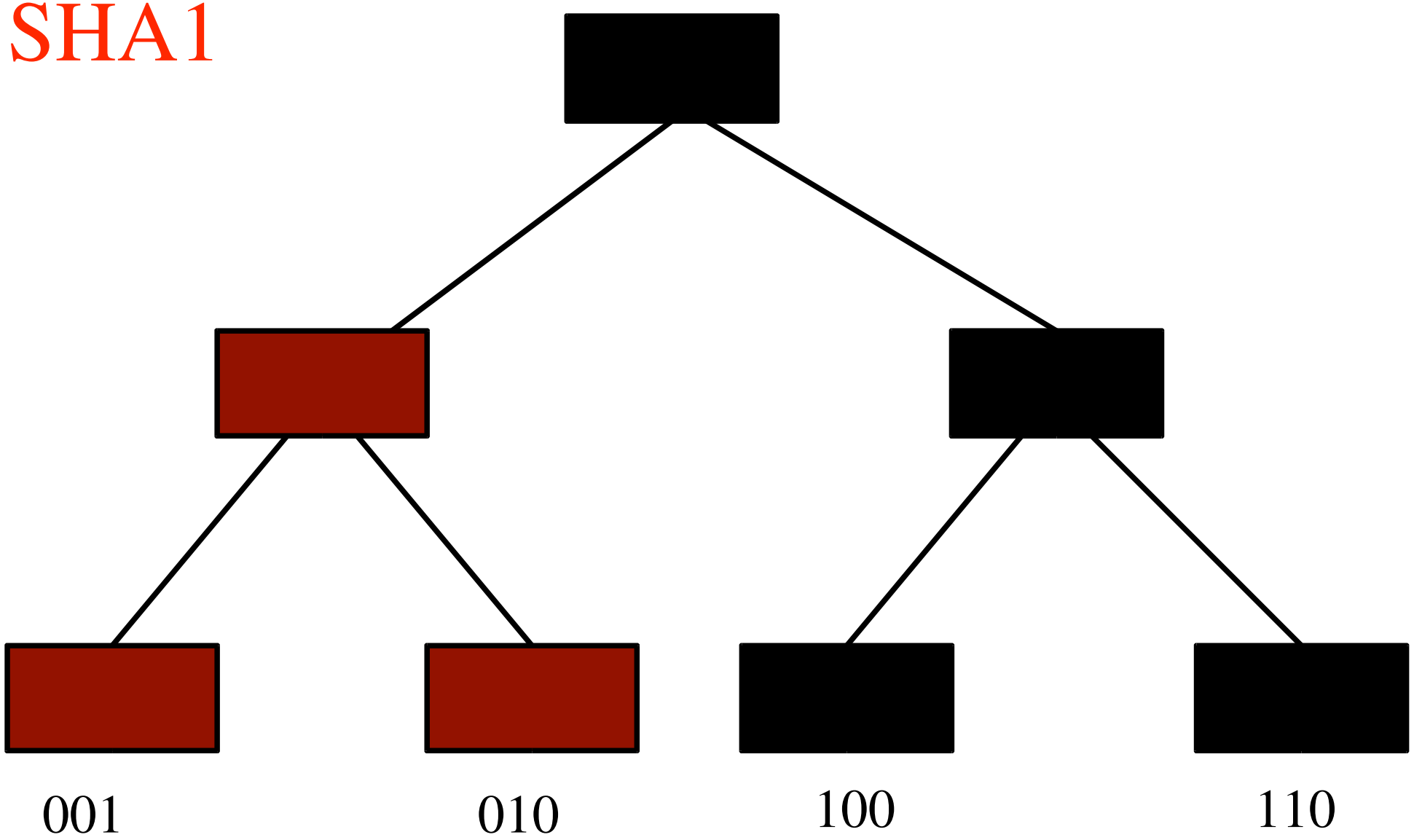
Digression: merkle tries

SHA1



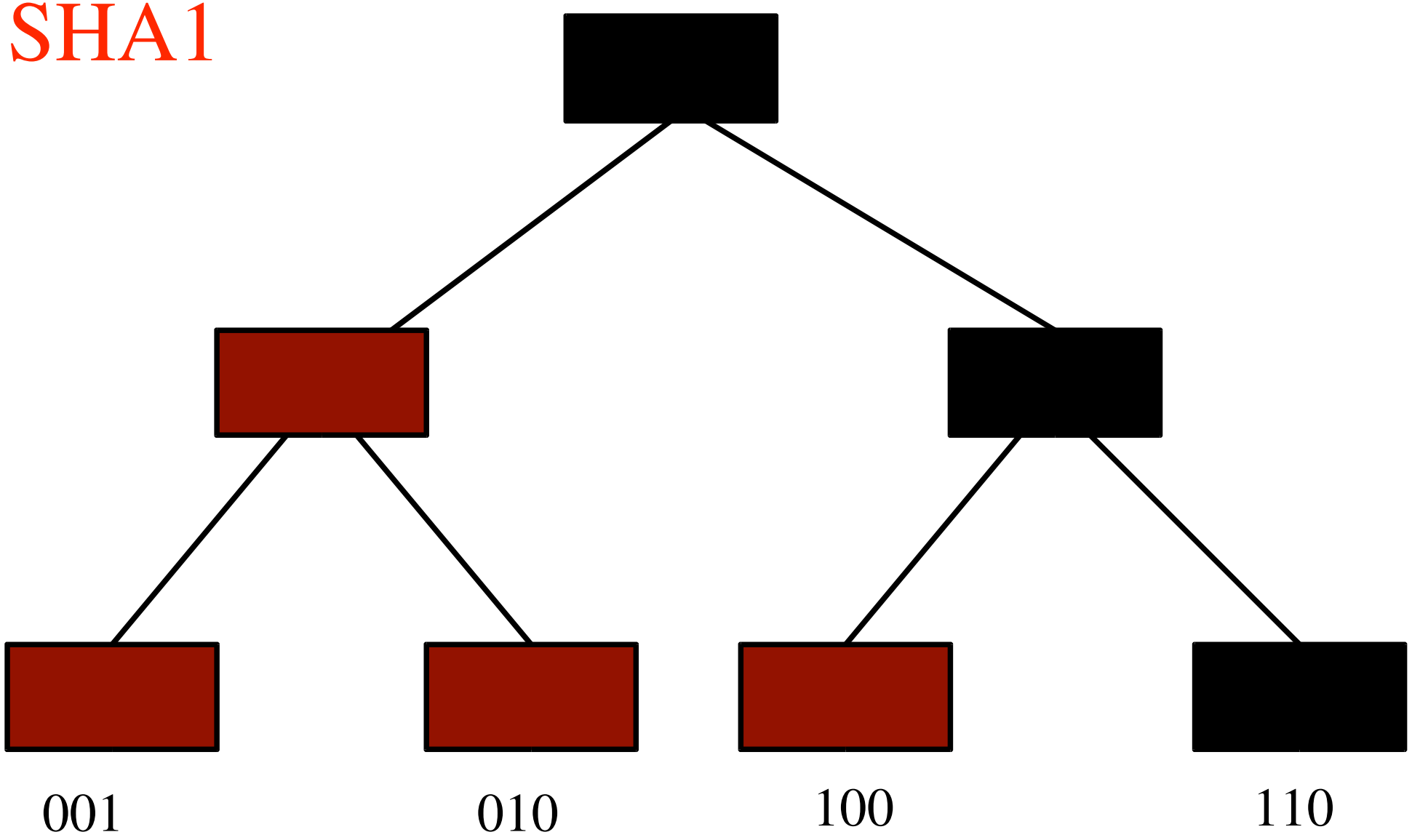
Digression: merkle tries

SHA1



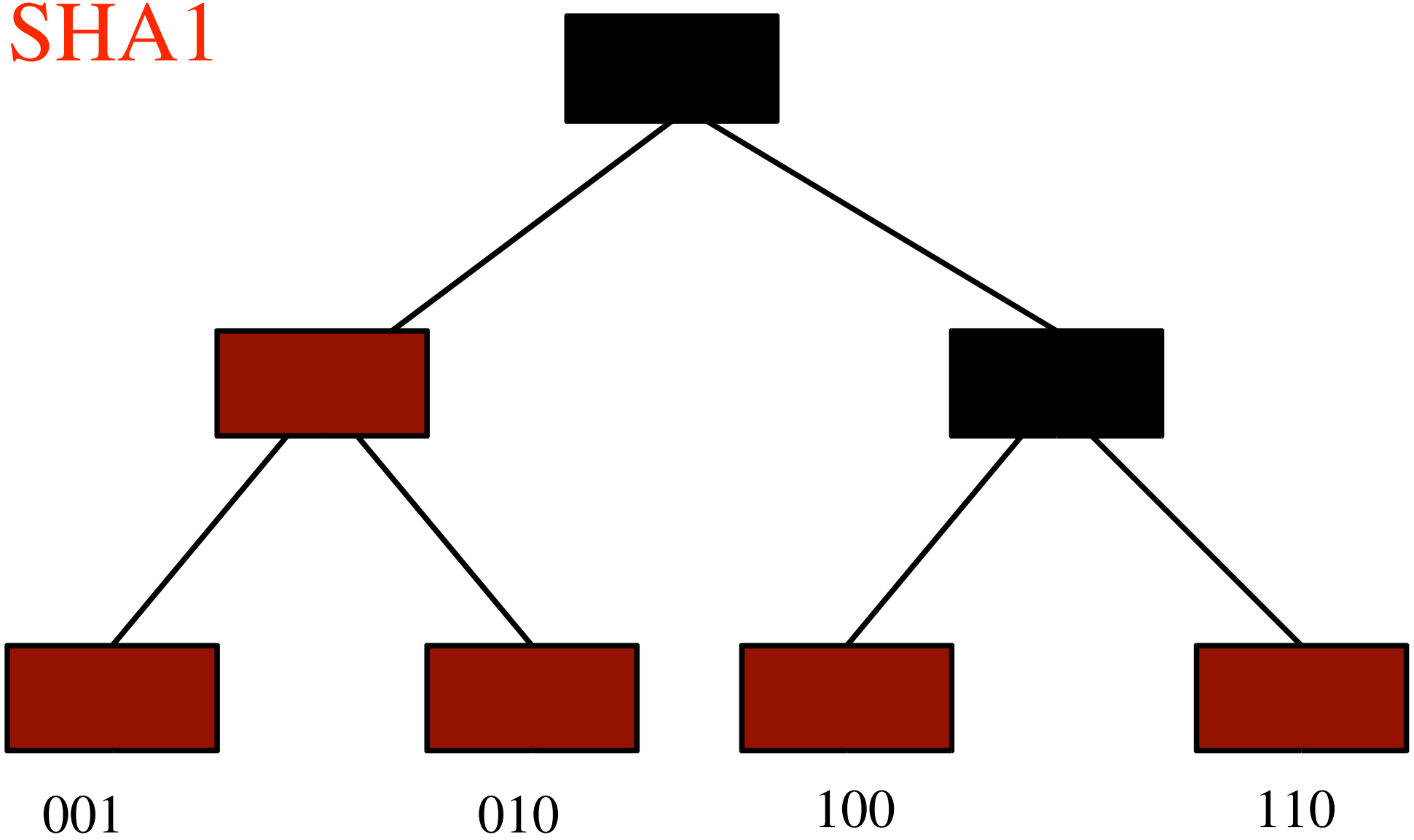
Digression: merkle tries

SHA1



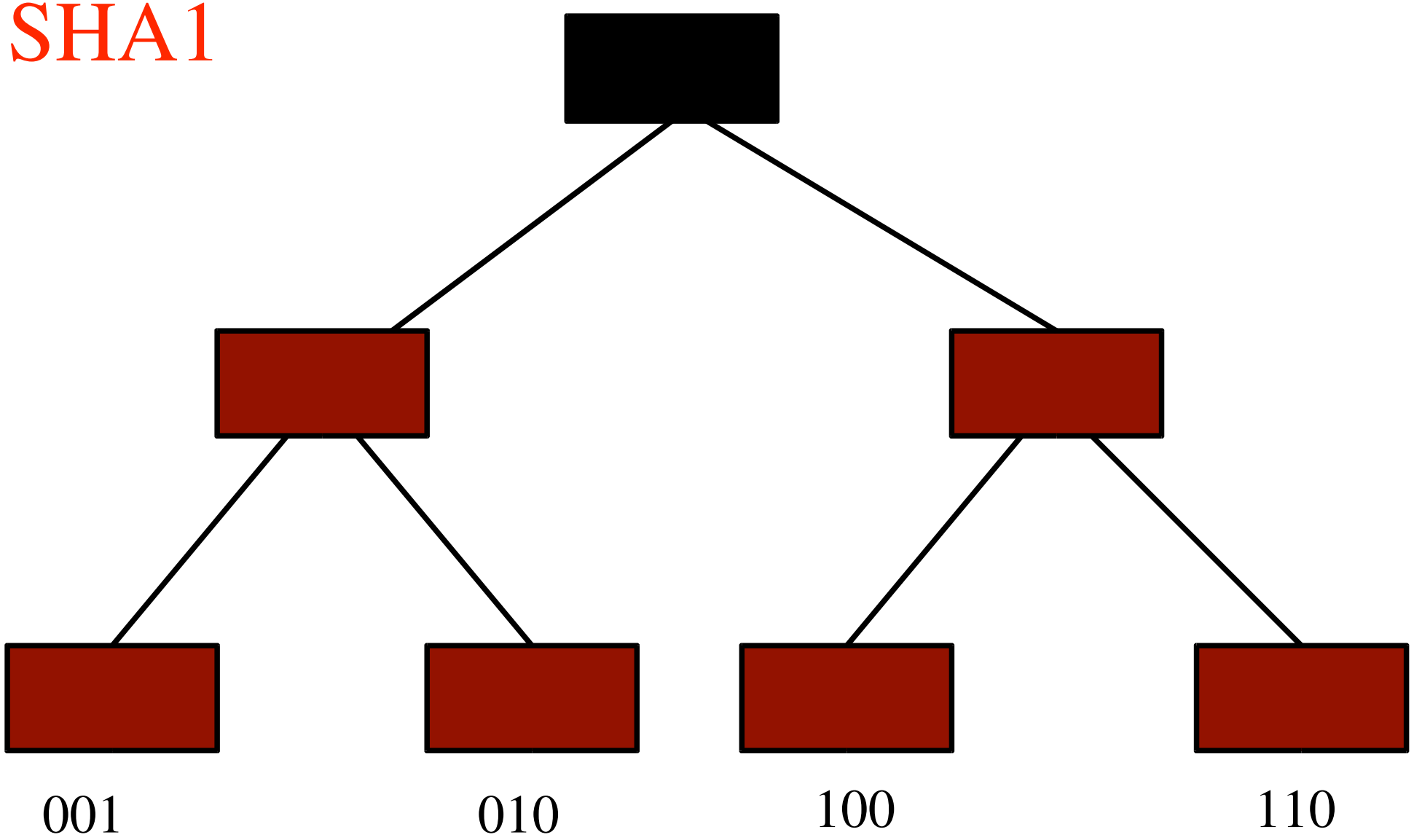
Digression: merkle tries

SHA1



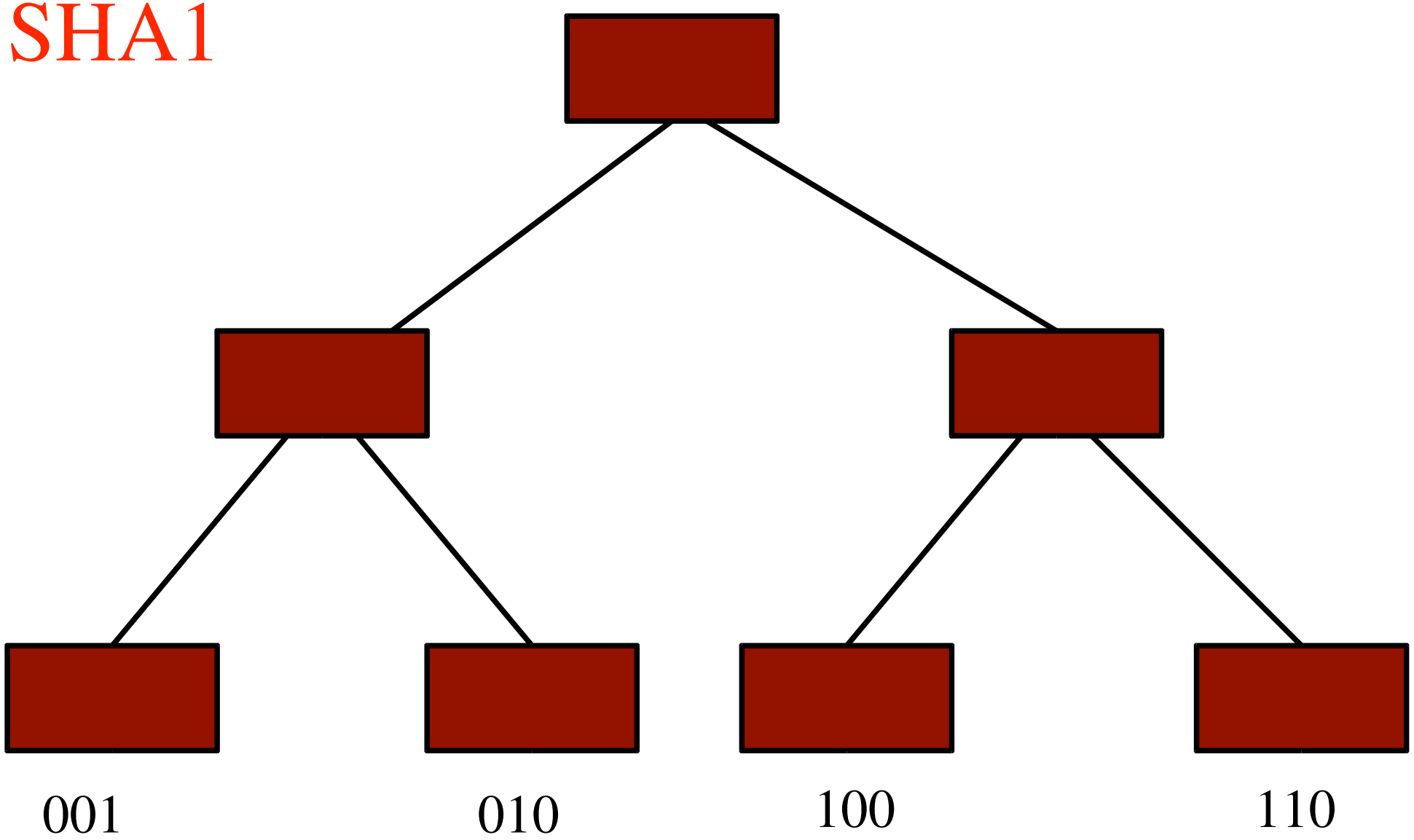
Digression: merkle tries

SHA1



Digression: merkle tries

SHA1



Digression: merkle tries

Arbitrary set synchronization

Pipelining friendly

O(d log n) bytes, (log n)/2 round trips

- where d is the size of the difference
- n is size of the overall set

Digression: merkle tries

Arbitrary set synchronization

Pipelining friendly

O(d log n) bytes, (log n)/2 round trips

- where d is the size of the difference
- n is size of the overall set

(rsync scales as O(n))

I remain stress-free because...

I can understand it

Convinced?

I remain stress-free because...

I can understand it

Convinced?

(The implications are highly non-obvious.
Hence, the rest of the talk.)

I remain stress-free because...

I trust that it works right

I remain stress-free because...

I trust that it works right

...duh?

I remain stress-free because...

I trust that it works right

It's not – does it work?
It's – how do I know it works?

Including in the situations I haven't used it in yet?

I remain stress-free because...

I trust that it works right

Moral:

Use software written by crazy paranoid people.


INSTANT
krazy
GLUE
ADVANCED
FORMULA
EXTRA
STRENGTH
GEL
• SHOCK RESISTANT BOND
• EXTRA STRENGTH AND DURABILITY FOR POTTERY, CERAMICS AND OTHER POROUS SURFACES
• NO-CLOG PRECISION APPLICATOR
WARNING: EYE IRRITANT
BONDS SKIN INSTANTLY
NET WT 0.14 OZ (4g)

I remain stress-free because...

I trust that it works right

Instrumentation:

-- logging (always on)

-- stack and data tracing

-- all are dumped to file on crash

Regularly fix problems that are non-reproducible, occurred in the field, on repositories we have no access to.

I remain stress-free because...

I trust that it works right

Wait... “crash”?

I remain stress-free because...

I trust that it works right

Wait... “crash”?

-- monotone is a C++ program that does not segfault

-- 4 kinds of assertions

-- crash only, crash early

-- logging and assertions are single characters, to maximize use

I remain stress-free because...

I trust that it works right

Some quick statistics:

-- total executable lines: ~16000

-- assertions: ~850 (1 in ~20 lines)

-- logging: ~670 (1 in ~24)

-- data instrumentation: 253 (1 in ~64)

Total: 1 in ~9 lines devoted to error detection and diagnosis

I remain stress-free because...

I trust that it works right

Development process:

-- 90% test coverage

-- continuous build/test on 10 boxes, 5 operating systems, 4 architectures (we want more!)

-- coverage information also generated continually and linked from front of web site

I remain stress-free because...

I trust that it works right

Coding style:

-- No pointers

-- Almost no explicit heap allocation

-- Extreme use of type system

It's in C++ entirely for the type system.

I remain stress-free because...

I trust that it works right

The compiler will reject:

-- code that would allow a path to escape the working copy

-- code that passes a hash of a file where we wanted the hash of a revision

-- gzipped-but-not-base64'ed data to a function that wanted gzipped-and-base64'ed data

I remain stress-free because...

I trust that it works right

Higher level – successful robustness must be baked in to the architecture.

I remain stress-free because...

I trust that it works right

Examples:

-- using hashes as names effectively tunnels strong end-to-end security over existing, social channels (IRC, mailing lists, post-it notes...)

-- 'sync' keeps no state about peers, therefore cannot have bugs related to state tracking

I remain stress-free because...

I trust that it works right

See how many more you notice...

I remain stress-free because...

I never worry about my data

-- there is never any reason not to sync changes out; it is a safe operation

-- sync always pushes all of my changes, and pulls all of everybody else's changes

--> every change is backed up on every developer's computer

I remain stress-free because...

I never worry about my data

---> "restore from backup" is *the same command and code paths* as I use all day, every day

I remain stress-free because...

I never worry about my data

Self-imposed rule:

If we store a piece of information,
we must verify that piece of information.

Optimization problem: data structures that can be efficiently verified.

I remain stress-free because...

I never worry about my data

When pulling data, every piece is verified (hashes, well formedness, semantic consistency)
When reading database (checkout, update, ...), data is always verified before the user can see it.

Monotone worries, so you don't have to.

I remain stress-free because...

I never worry about my data

But that must be so slow!!!

I remain stress-free because...

I never worry about my data

Sneaky trick – during an initial pull is the only time a VCS can operate on the whole database.

Doing exhaustive checking here means that we have early detection of server corruption that only affects old, never used versions...

I remain stress-free because...

I never worry about my data

'db check'

I remain stress-free because...

I never worry about my data

Monotone has been self-hosting, using its latest bleeding edge, continuously since September 2003.

I remain stress-free because...

I never worry about my data

Monotone has been self-hosting, using its latest bleeding edge, continuously since September 2003.

Other projects have tens of thousands of revisions stored.

I remain stress-free because...

I never worry about my data

Monotone has been self-hosting, using its latest bleeding edge, continuously since September 2003.

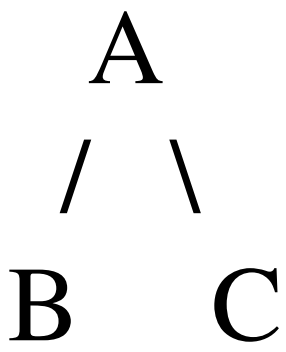
Other projects have tens of thousands of revisions stored.

No data stored in monotone has ever been lost. (That we know about.)

I remain stress-free because...

I can always see what happened

Consider three revisions:

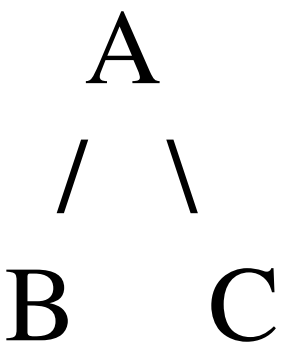


All have branch=Foo certs on them.

I remain stress-free because...

I can always see what happened

Consider three revisions:



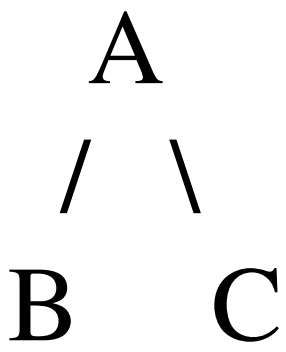
All have branch=Foo certs on them.

---> branch Foo has 2 heads?!?

I remain stress-free because...

I can always see what happened

Consider a checked in tree A, and two checkouts B and C:



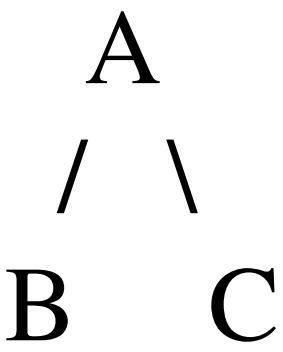
All are working on the same branch.

---> this branch has parallel work in it?!?

I remain stress-free because...

I can always see what happened

Consider a checked in tree A, and two checkouts B and C:



All are working on the same branch.

---> this branch has parallel work in it?!?

---> CVS, SVN say: throw it away!

I remain stress-free because...

I can always see what happened

Mini-demo

I remain stress-free because...

I can always see what happened

Moral:

parallelism exists

we can record it, or throw it away

I'd rather record it

--> one branch may have multiple heads

I remain stress-free because...

I don't have to think too hard

All other DVCSes:

-- branch = location (e.g., host+path)

-- copy --> create a *new*, *distinct* branch

I remain stress-free because...

I don't have to think too hard

Some things about locations:

-- to make a branch I have to set up a new location

-- no-one will know how to find my new branch unless I tell them

-- they have lots of these to keep track of, so maybe they'll remember, maybe not...

I remain stress-free because...

I don't have to think too hard

Some more things about locations:

-- each one has different rules for access

-- I can't automatically start hacking on my friend's branch

-- branches can disappear, so you need to mirror them...

-- quick, which mirrors do you update before getting on an airplane?

-- if a branch dies, who has a mirror?

I remain stress-free because...

I don't have to think too hard

Option 1: start adding machinery to solve each of these problems

Option 2: don't make the problems in the first place

I remain stress-free because...

I don't have to think too hard

Option 1: start adding machinery to solve each of these problems

Option 2: don't make the problems in the first place

I remain stress-free because...

I don't have to think too hard

Monotone:

-- locations are ephemeral and carry no state

-- every copy is a peer, no distinction between “original” and “mirror”

-- so... since we don't need to track anything, can just throw it all together

I remain stress-free because...

I don't have to think too hard

Result:

-- adding a branch is trivial and involves no administrative work

-- everyone sees all branches, because they are all mirrored on the group's server

-- shared branches are the default

-- everyone mirrors everything

I remain stress-free because...

commit and push always work

In a location-based system, communication is a mutating operation. In monotone, communication is purely information.

Thus, monotone commit and push always work and are safe.

In other systems, one or both may necessarily involve a merge.

I remain stress-free because...

commit and push always work

Scenario:

You're in a hurry. You need to catch a plane. Or your battery is about to die. Or you want to clock out and go home. Or your hard drive is warning you it will catch fire any moment now.

You have some finished work, and you want to get it out of your working copy, and off your hard drive.

I remain stress-free because...

commit and push always work

```
$ vcs1 commit
error: working copy is out of date
$ vcs1 update
merging changes...
7 conflicts in 3 files

$ vcs2 commit
$ vcs2 push
attempting to merge...
encountered conflicts, pull first
```

I remain stress-free because...

commit and push always work

(in the latter case you could make push to a new branch, but perhaps you would not like your workflow so dictated...

in monotone we suggest that branches should mark communal purposes, not “some divergence happened”)

I remain stress-free because...

I can build the workflow I want

A VCS is part of an ecology of tools. Certs are designed to let you integrate with whatever you want – I don't know what all you can do! Some ideas:

-- tracking branch status (cf. Xaraya)
-- managing code review
-- tracking build/test results
-- linking to bug trackers
-- you tell me...

I remain stress-free because...

I can get my work done...

I remain stress-free because...

I can get my work done...

...and you can't stop me!

I remain stress-free because...

I can get my work done...

...and you can't stop me!

If someone breaks the build, route around them.

I remain stress-free because...

I can get my work done...

...and you can't stop me!

Don't update while in the middle of work; who needs conflicts then anyway? Commit first, then merge.

I remain stress-free because...

I can get my work done...

...and you can't stop me!

If you discover you're working against something already broken... update backwards to something that isn't! update will move your changes in any direction when requested, for exactly this reason.

I remain stress-free because...

I can get my work done...

...and you can't stop me!

Use testresult certs; your update commands will automatically ignore any broken builds.

Summary

We're good at:

- -- shared branches
- -- group awareness
- -- reducing friction in sharing and collaboration
- -- simple representations
- -- high tech:
  - -- first class directories
  - -- full support for renames, including directory renames, and merging
  - -- arbitrary file/directory attributes (with merger support)
  - -- only shipping implementation of a provably correct merge algorithm
- -- i18n'ized, available in 5 languages
- -- fully supported on Win32, OS X, BSD, Solaris, Linux
- -- crypto and end-to-end guarantees, in a friendly and transparent way
- -- crazy insane paranaoic approach to design and coding
  - -- self hosting since September 2003, with no recorded data loss by any project

Summary

Why not monotone?:

-- speed on initial pull (but stay tuned – we have fixed the second to last bug!)

-- several flag days between here and 1.0 (though migration is always provided).

-- currently requires every developer download a full copy

-- requires a dedicated server daemon

  -- though there is a design and prototype fixing this

-- UI polish is still in progress

-- lack of proper key/trust management, esp. 3rd party trust delegation

  -- we know how – ask for details if curious

I want...

...to understand.

...to have trust in the program.

...to never worry about my data.

...to record exactly what happened.

...safe commit and safe push.

...to make forward progress even when others screw up.

...to think about code. Not VC.