Teak: A token-flow implementation for Balsa

Andrew Bardsley
School of Computer Science
The University of Manchester

Balsa, Tangram and Haste

- The Balsa system generates Handshake Component/Circuit (HC) implementations of descriptions in the Balsa language
- Modelled on Tangram from Philips
- Tangram has become Haste/TiDE
- Balsa system includes a compiler, netlist generator, simulator, visualisation system

Balsa outings

- DMA controller for Amulet3 *
  - mixed sync/async design
- SPA - ARM in Balsa (G3CARD) *
  - Slow! A few MIPS in 180 nm
- nanoSpa - reworked SPA built for speed
- Silistix UART (Async 2008) ‘twig’

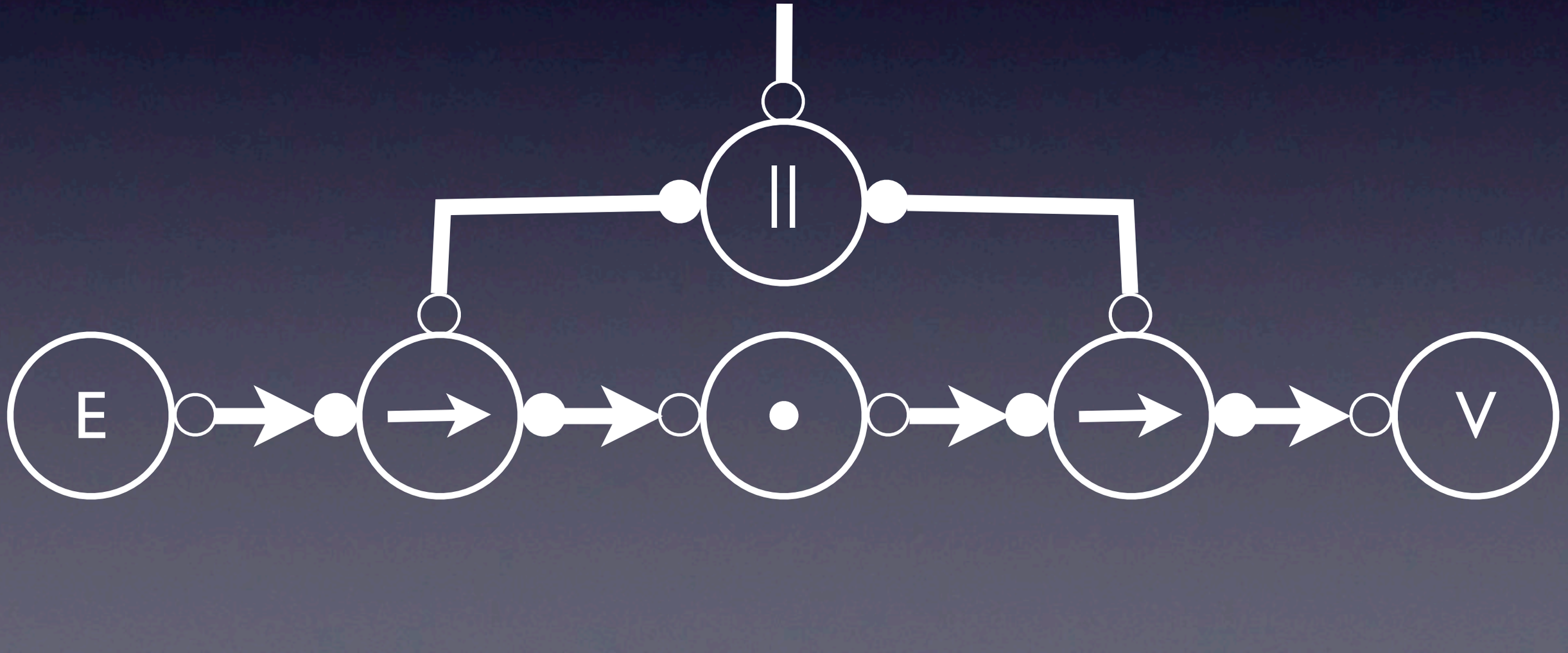
* in silicon

Improving Balsa

- Local efforts to increase Balsa performance:
  - Tibi Chelcea - burst-mode controller resynthesis
  - Luis Plana, Luis Tarazona - new FV components, compound control components (nanoSpa)
  - Sam Taylor - data-driven HCs. New language

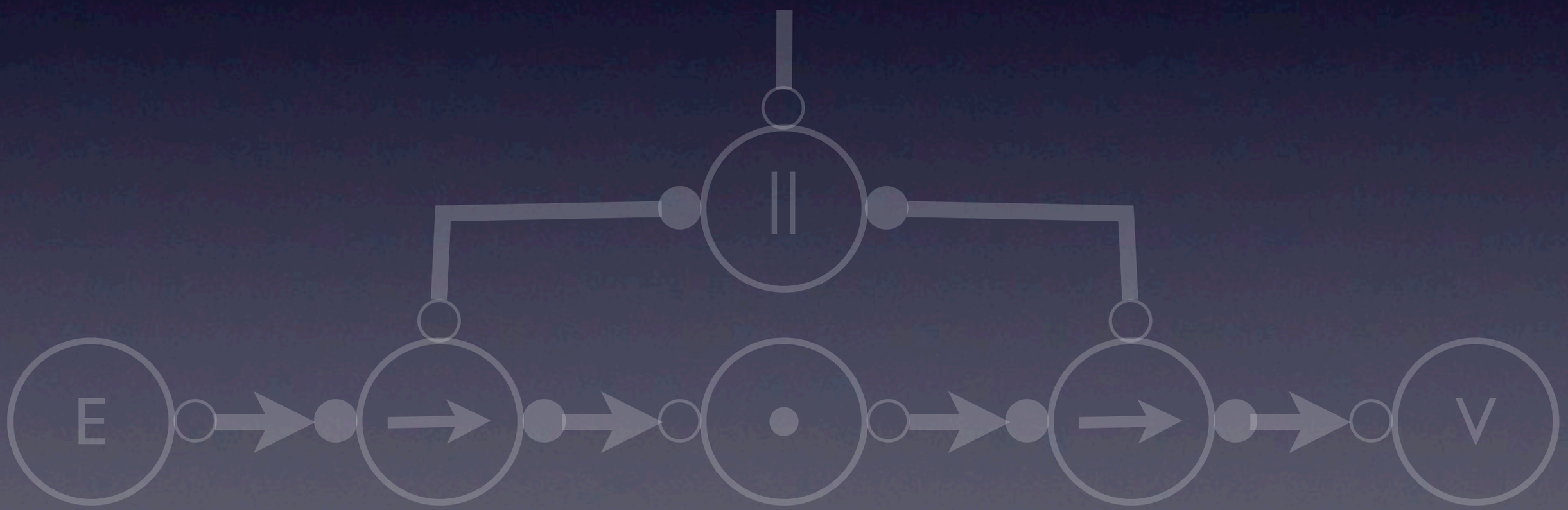
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



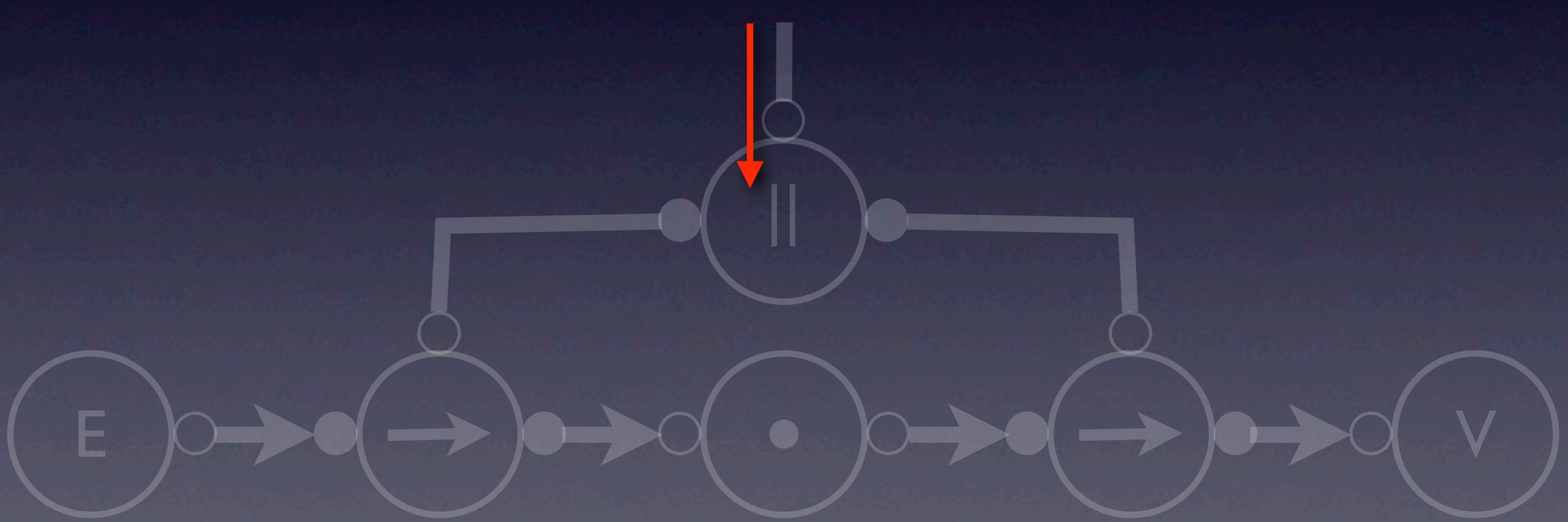
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



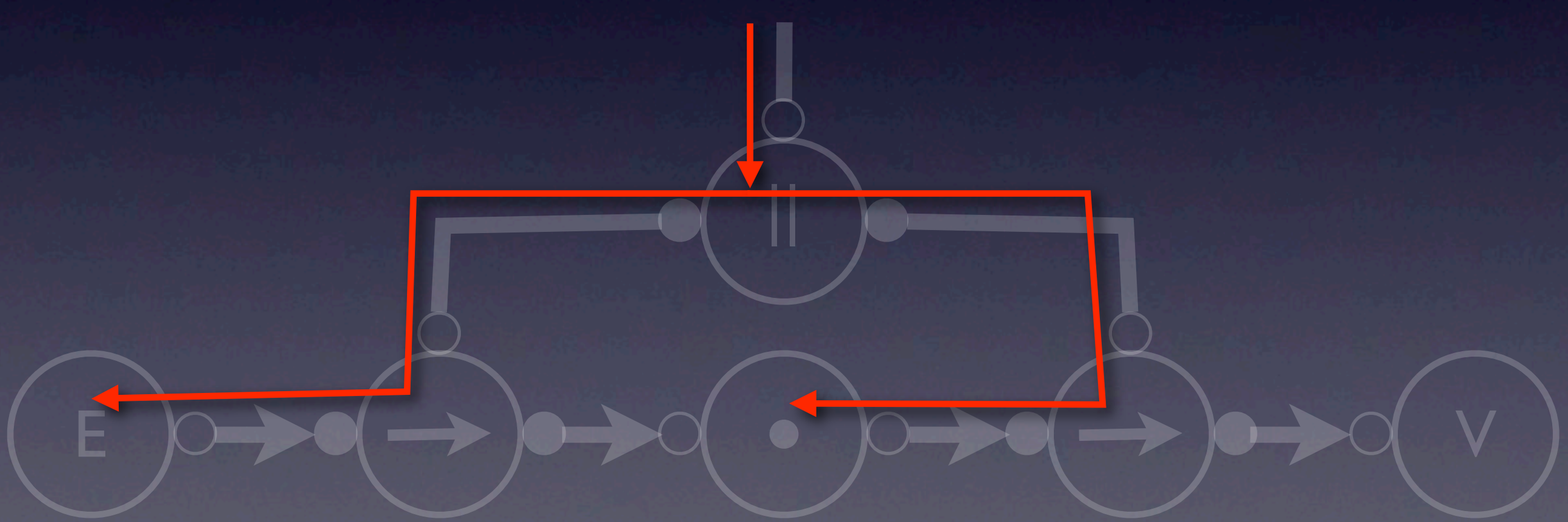
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



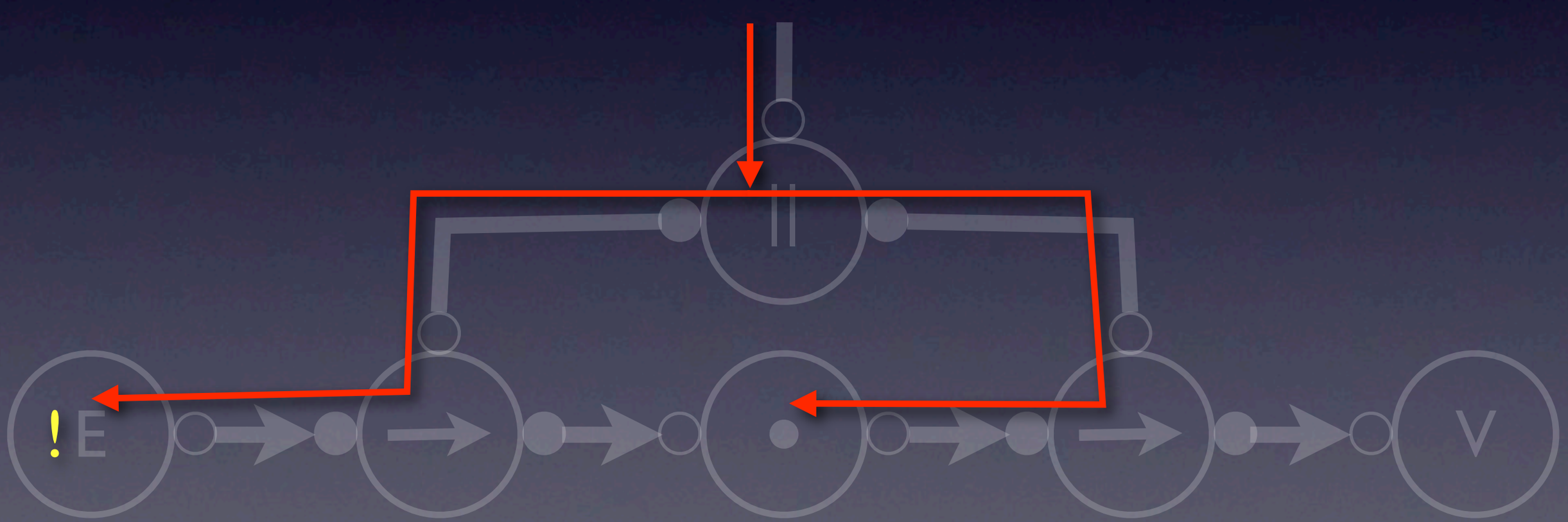
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



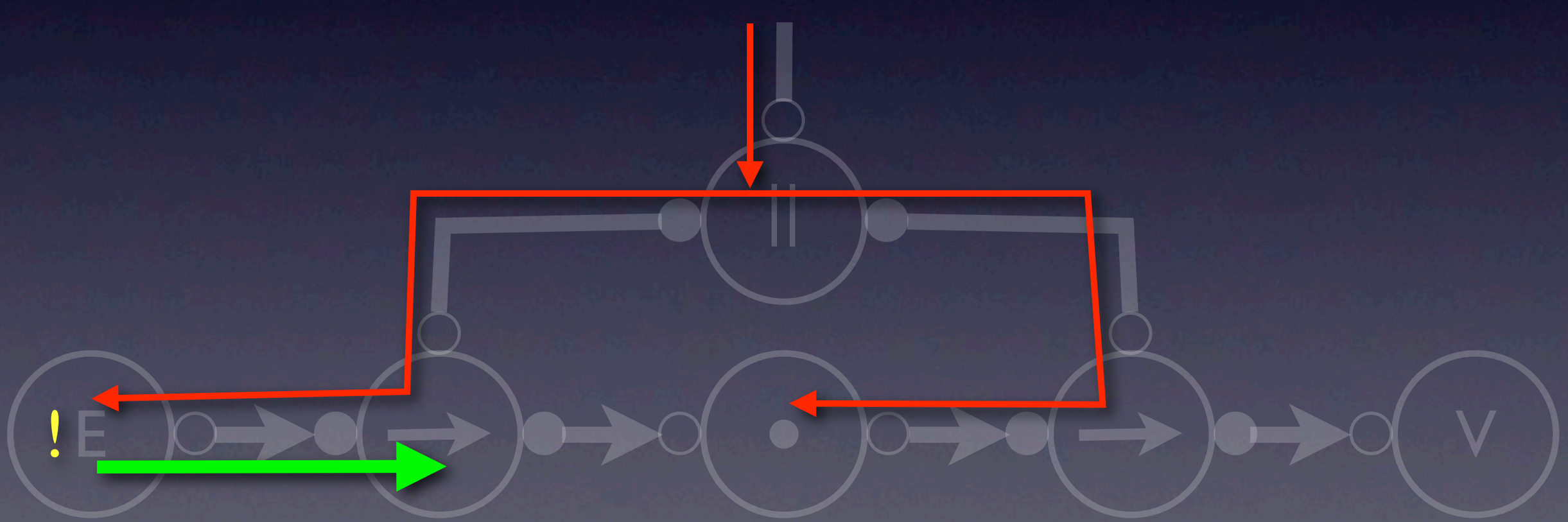
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



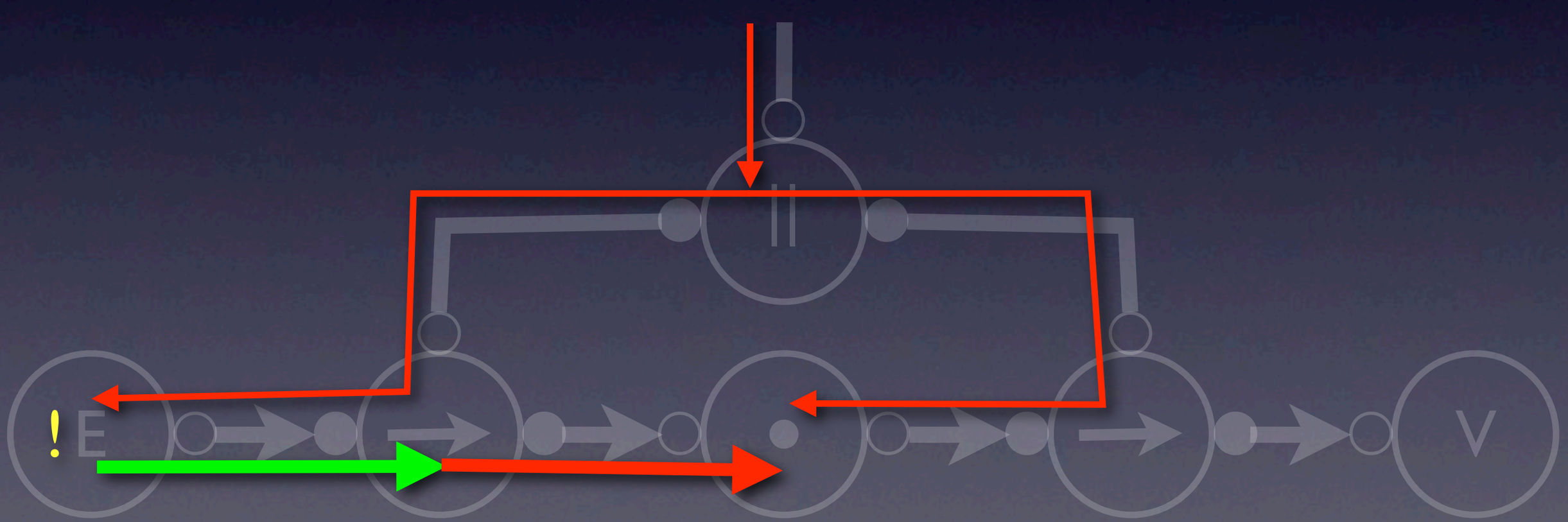
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



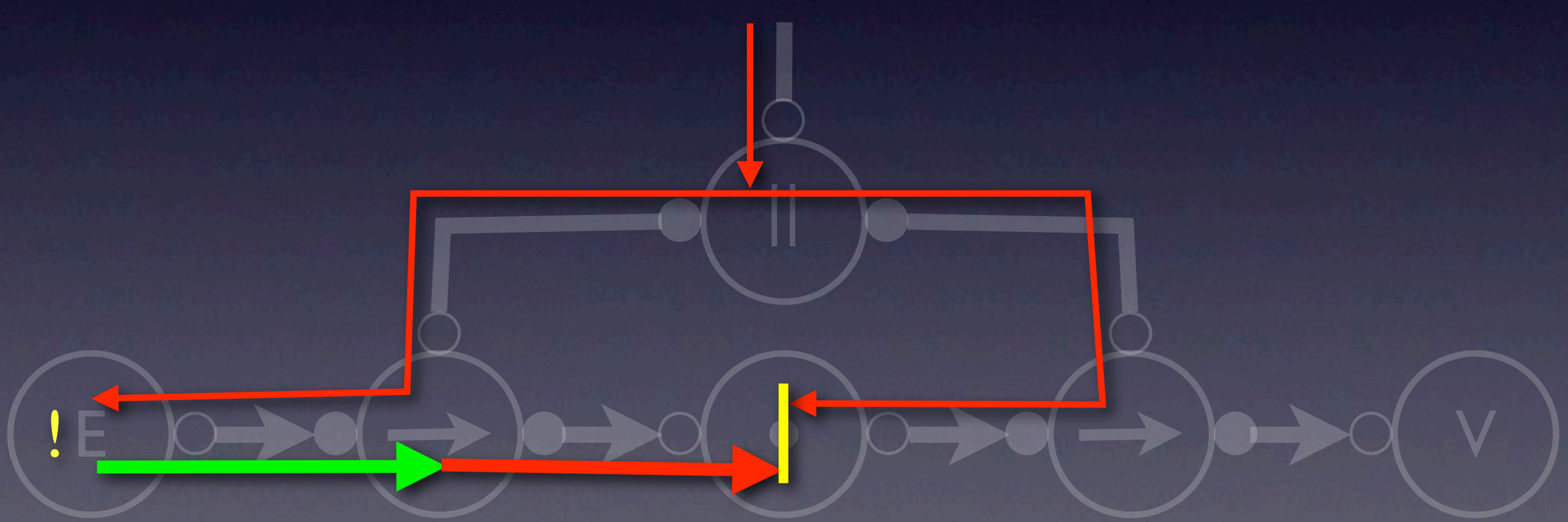
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



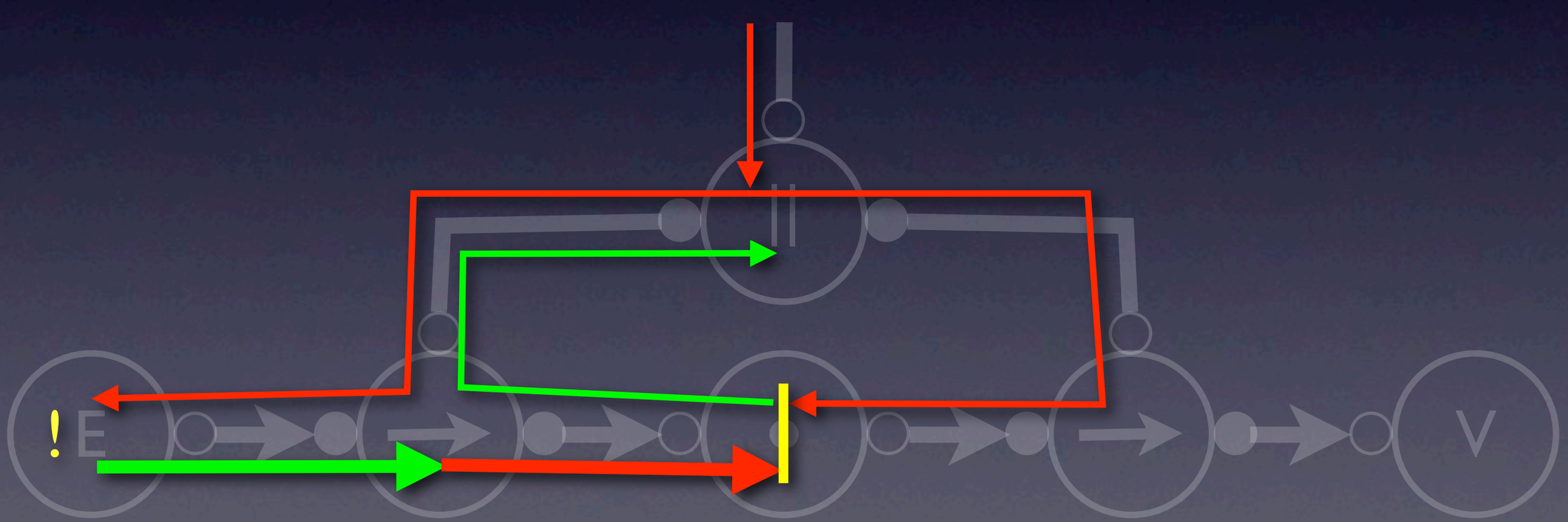
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



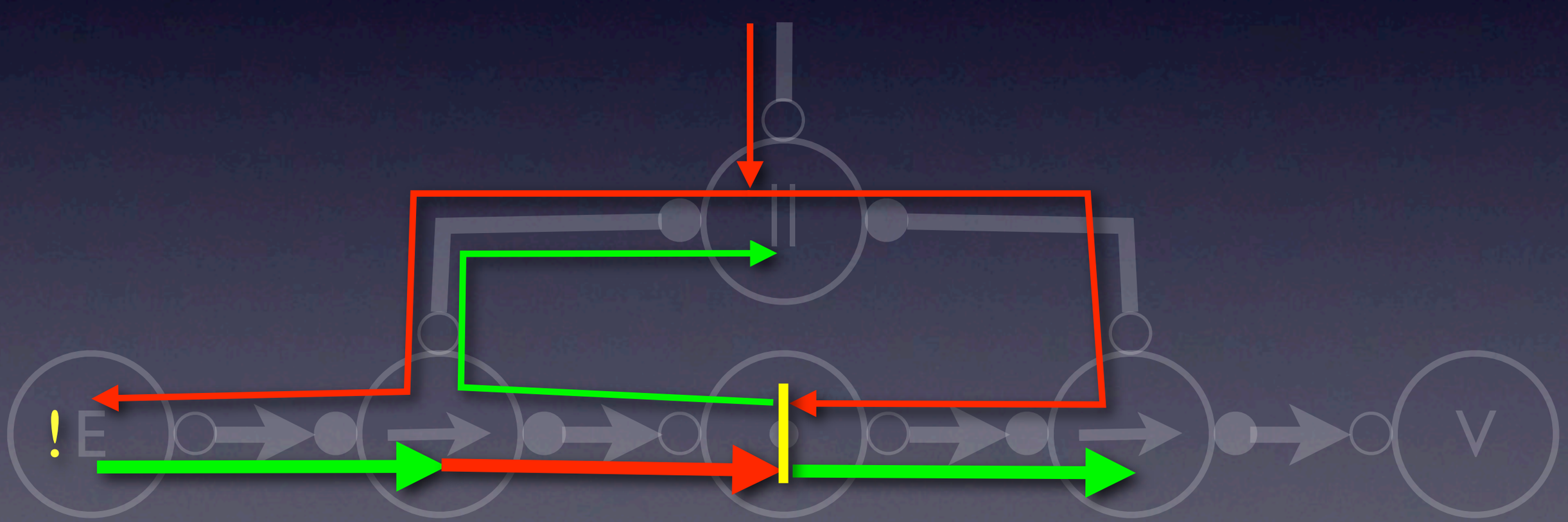
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



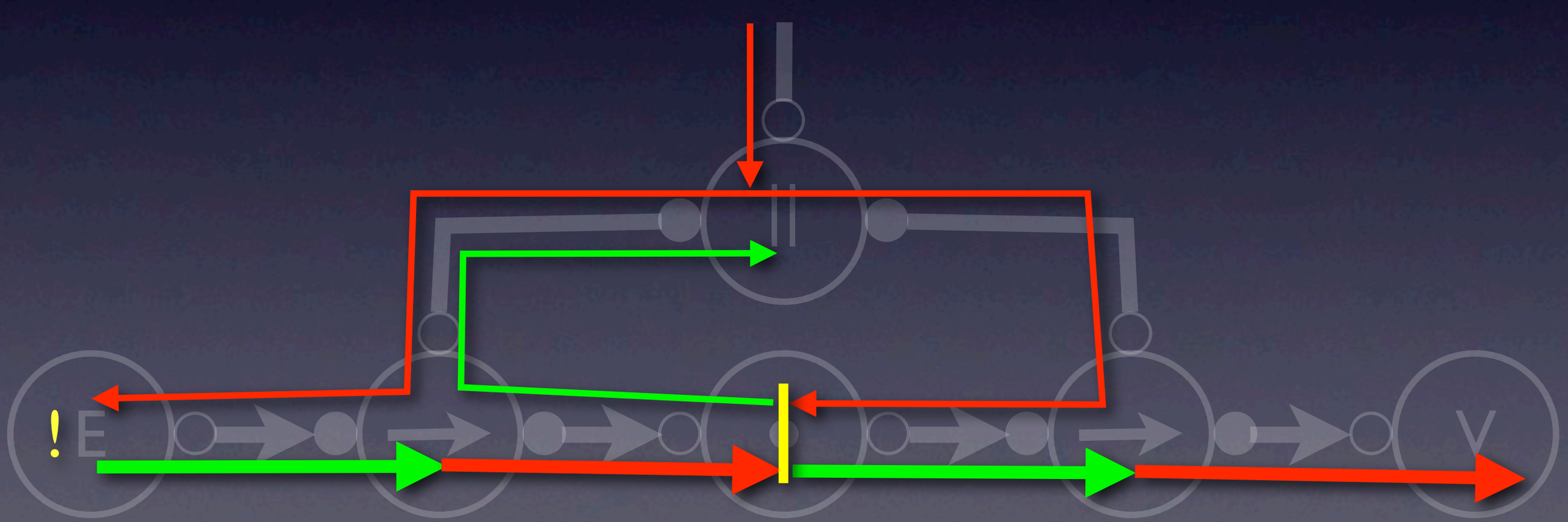
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



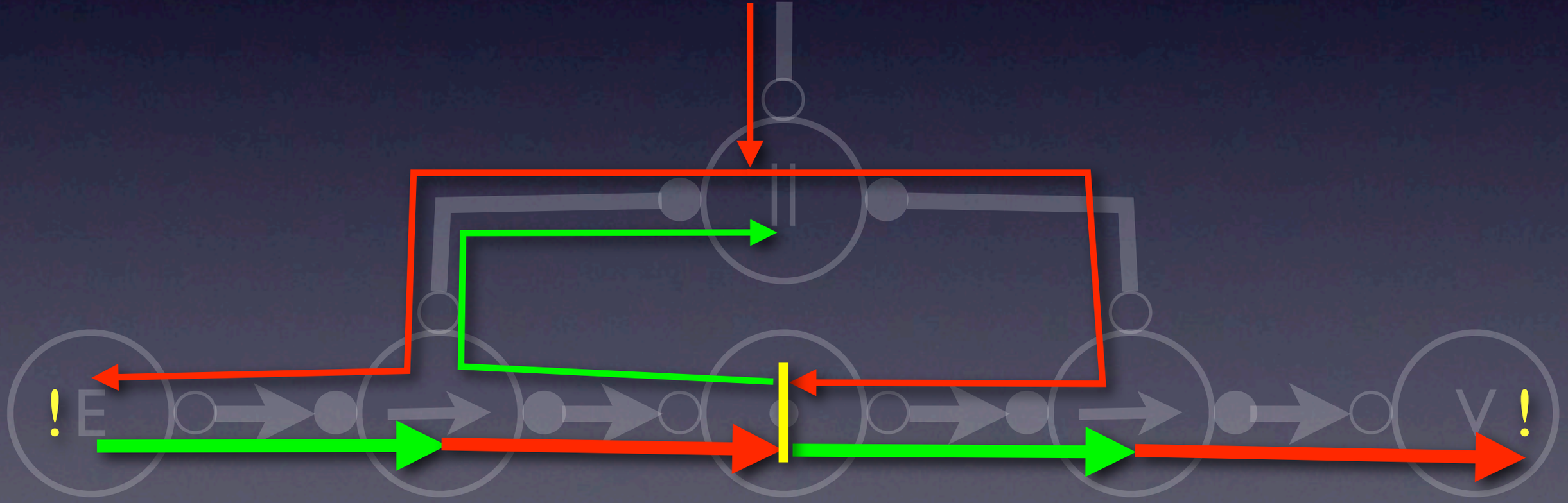
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



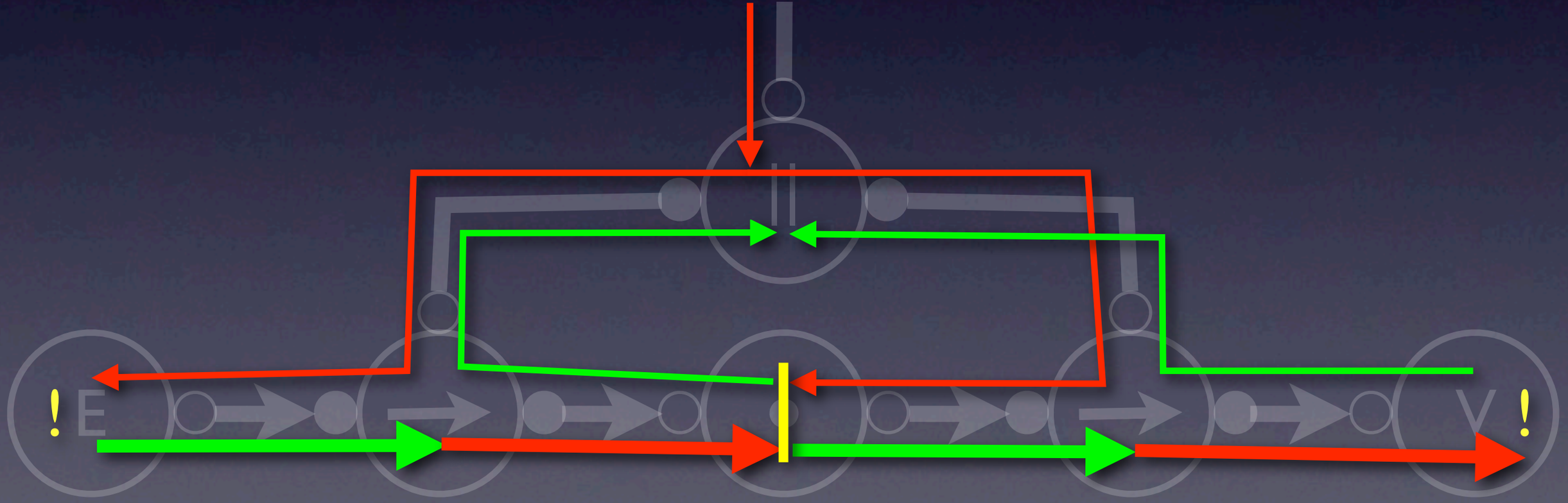
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



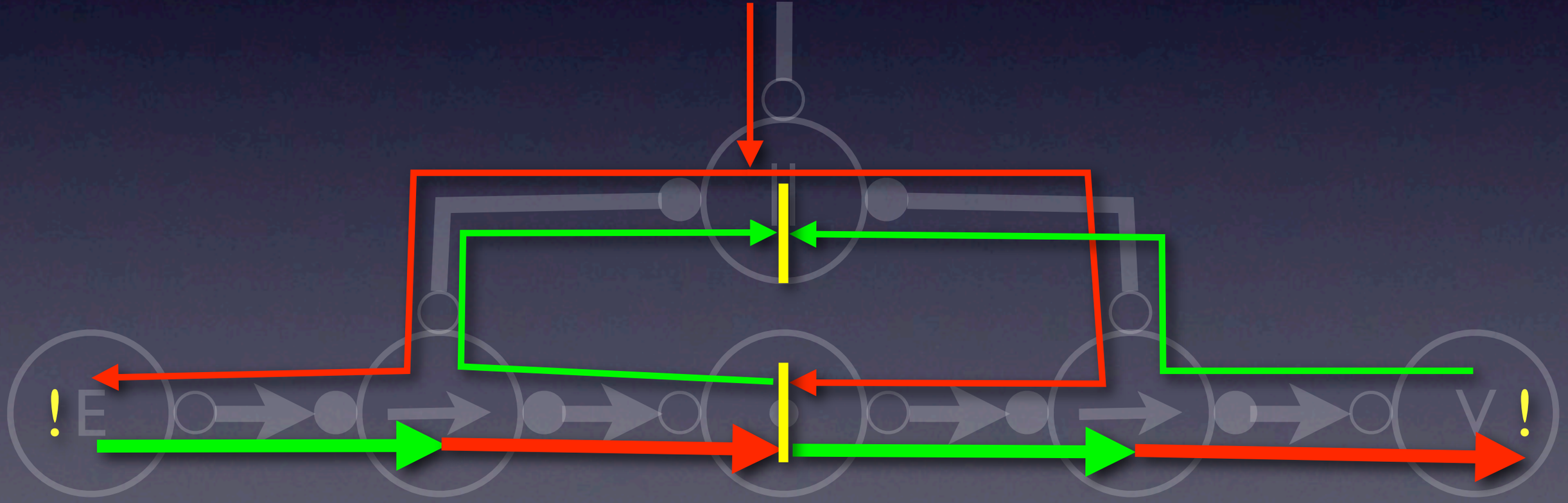
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



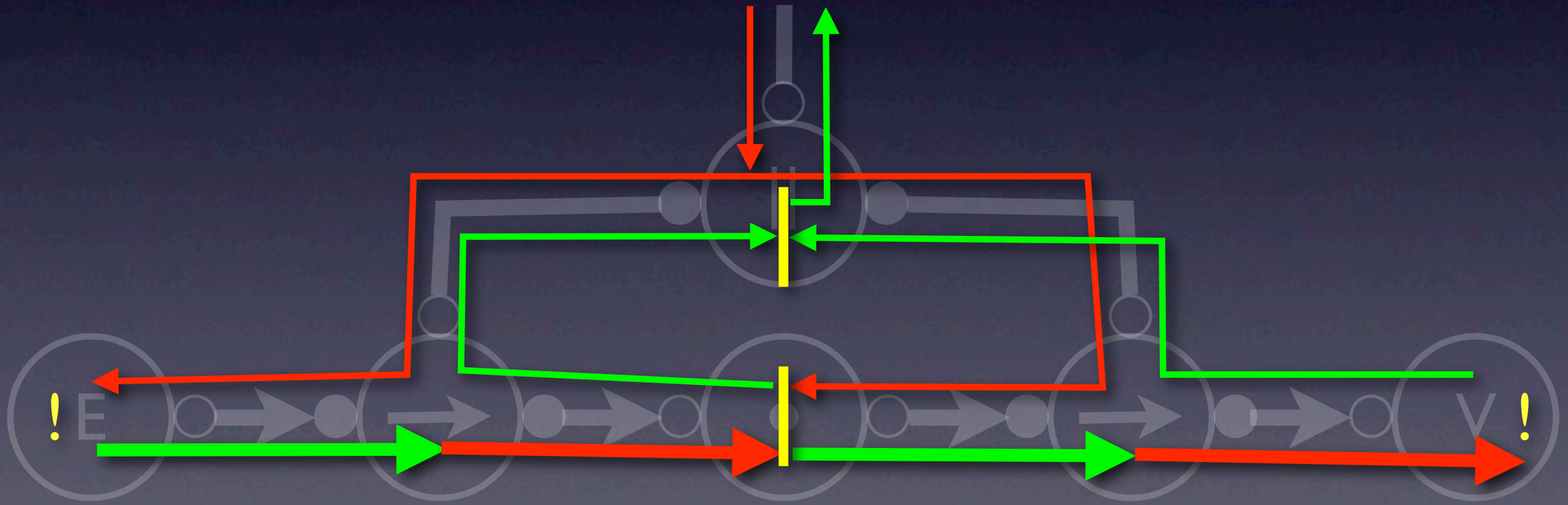
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



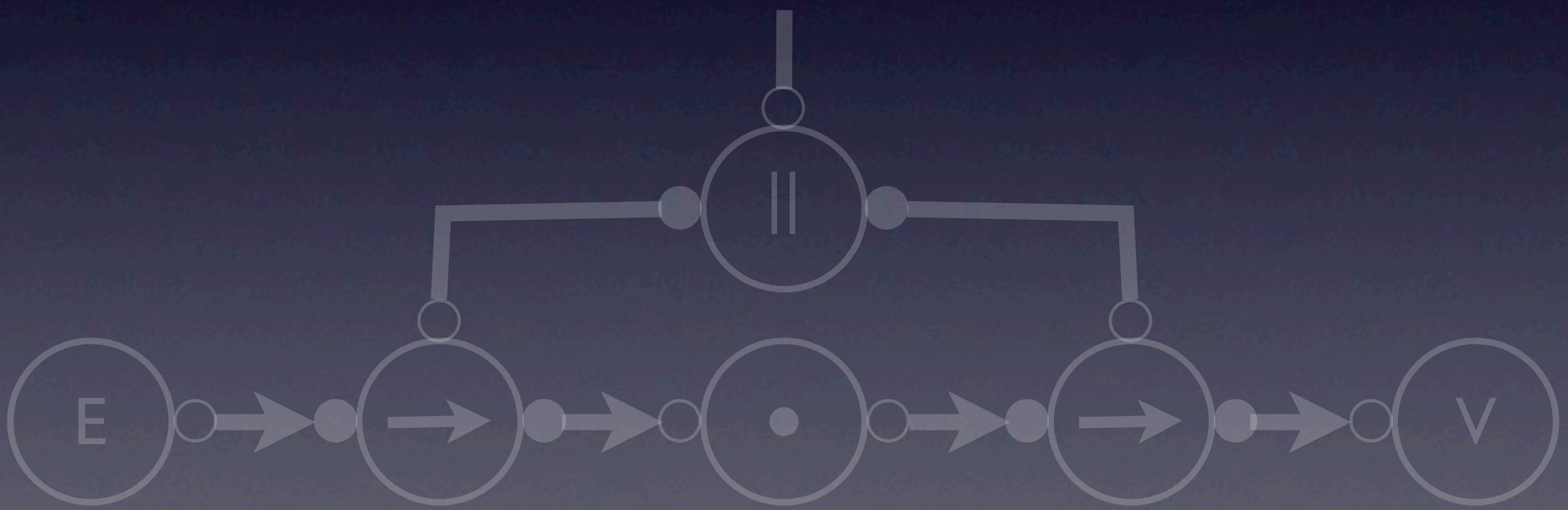
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



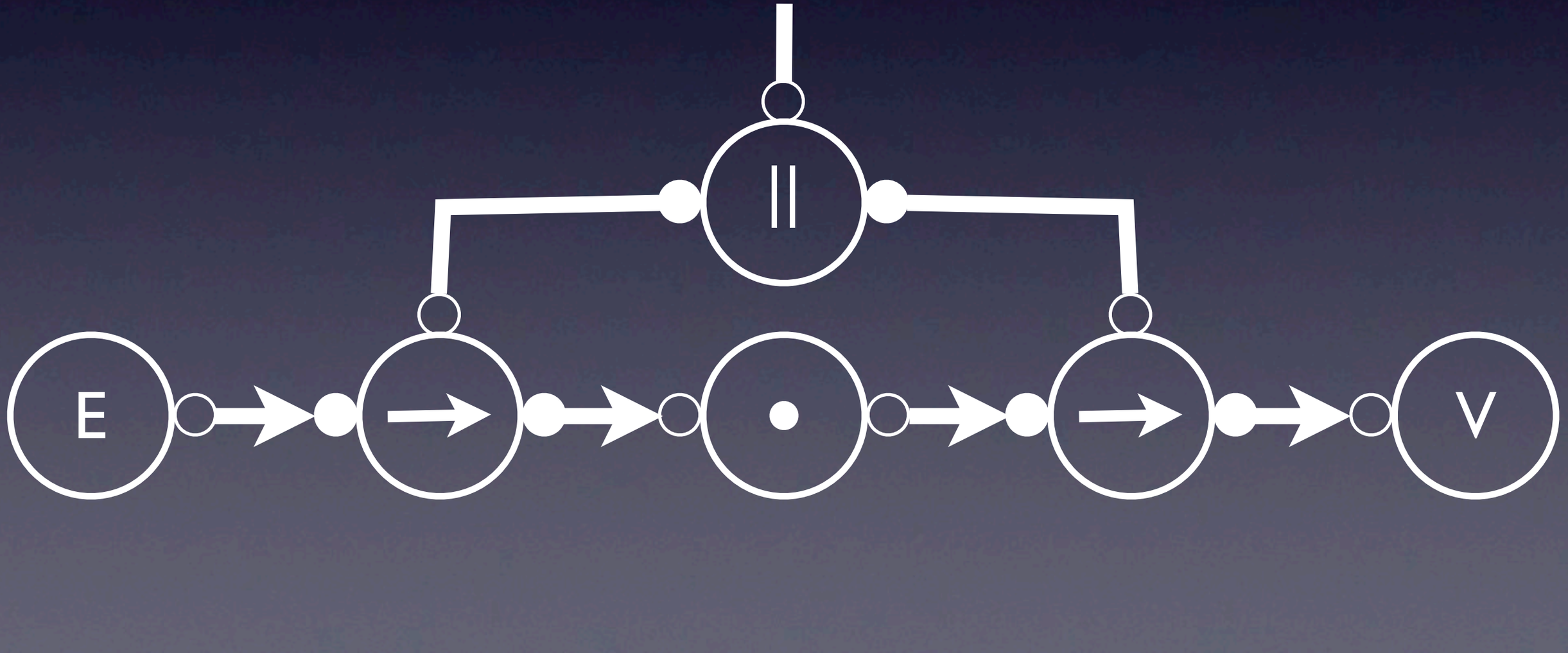
HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



HC compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`

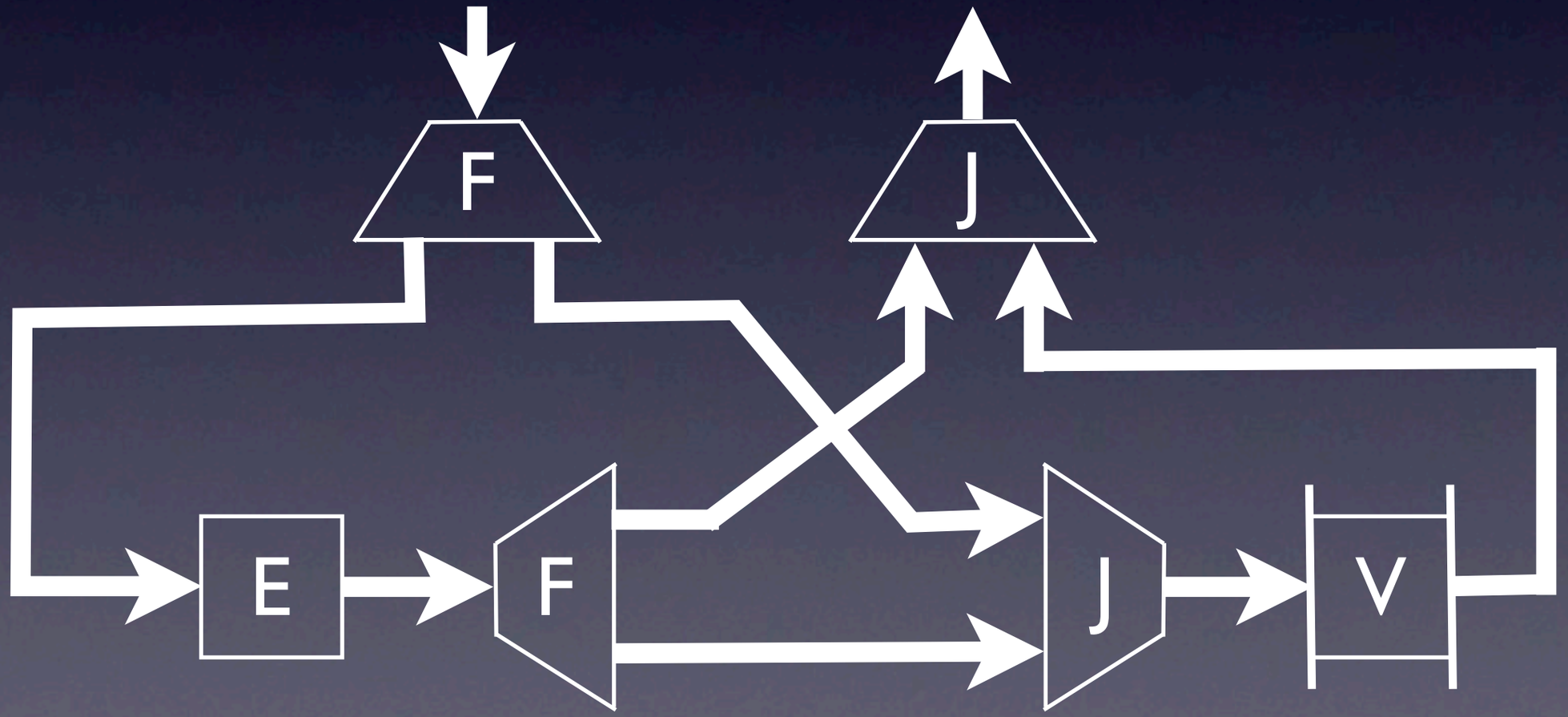


Why push stages?

- Reflects the way people want to write descriptions
- Reflects the way other implementation styles work - exploit other work more easily
- Enclosure flexibility - put your storage/ decoupling where you like
- Promise of concurrency

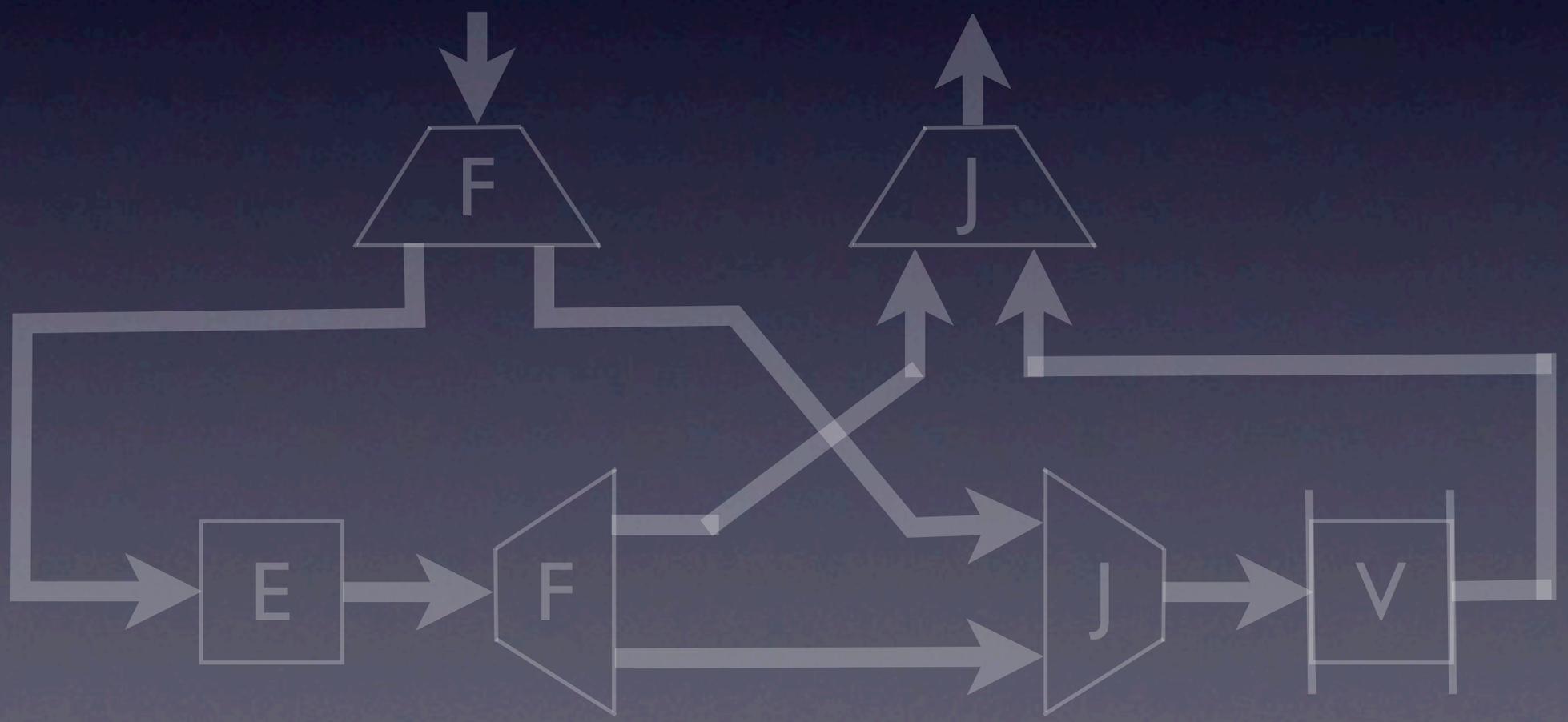
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



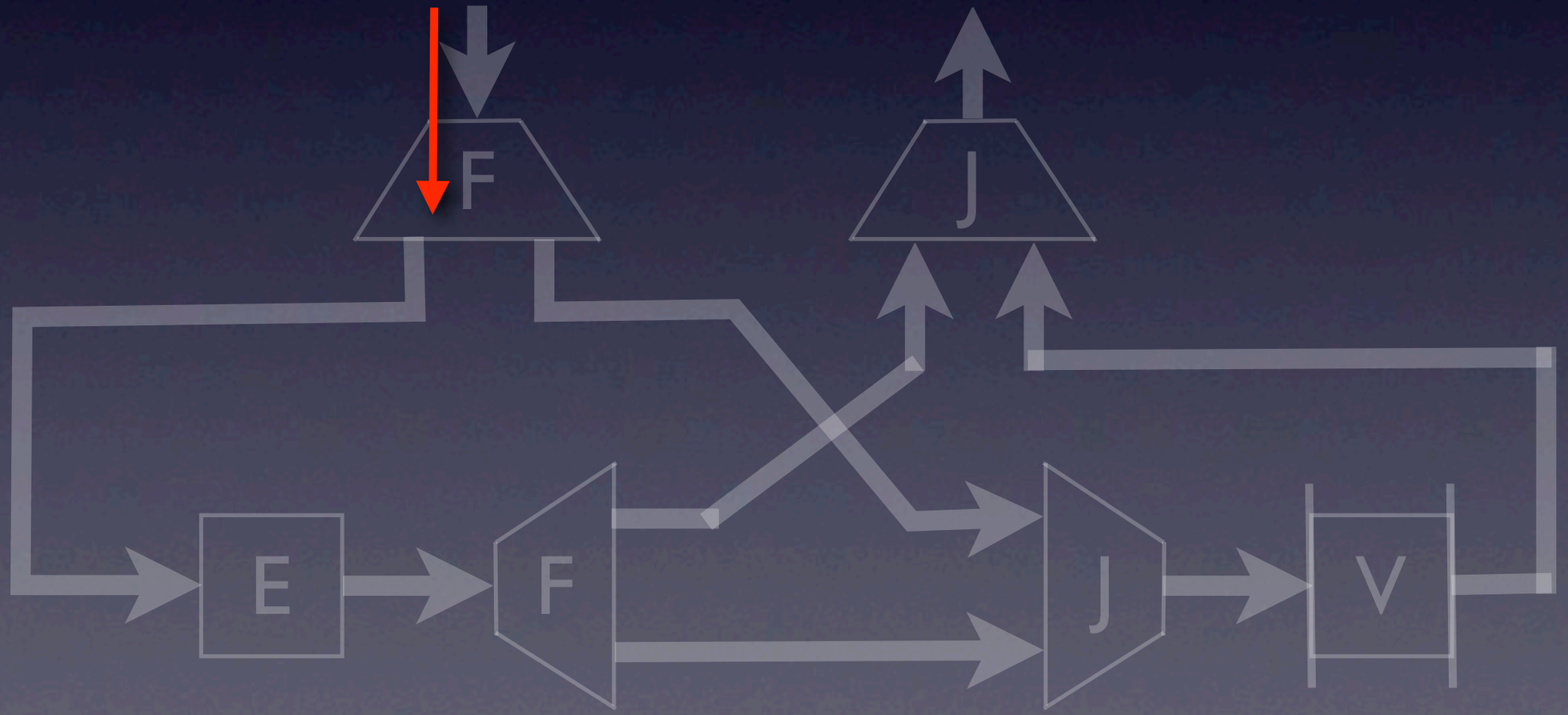
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



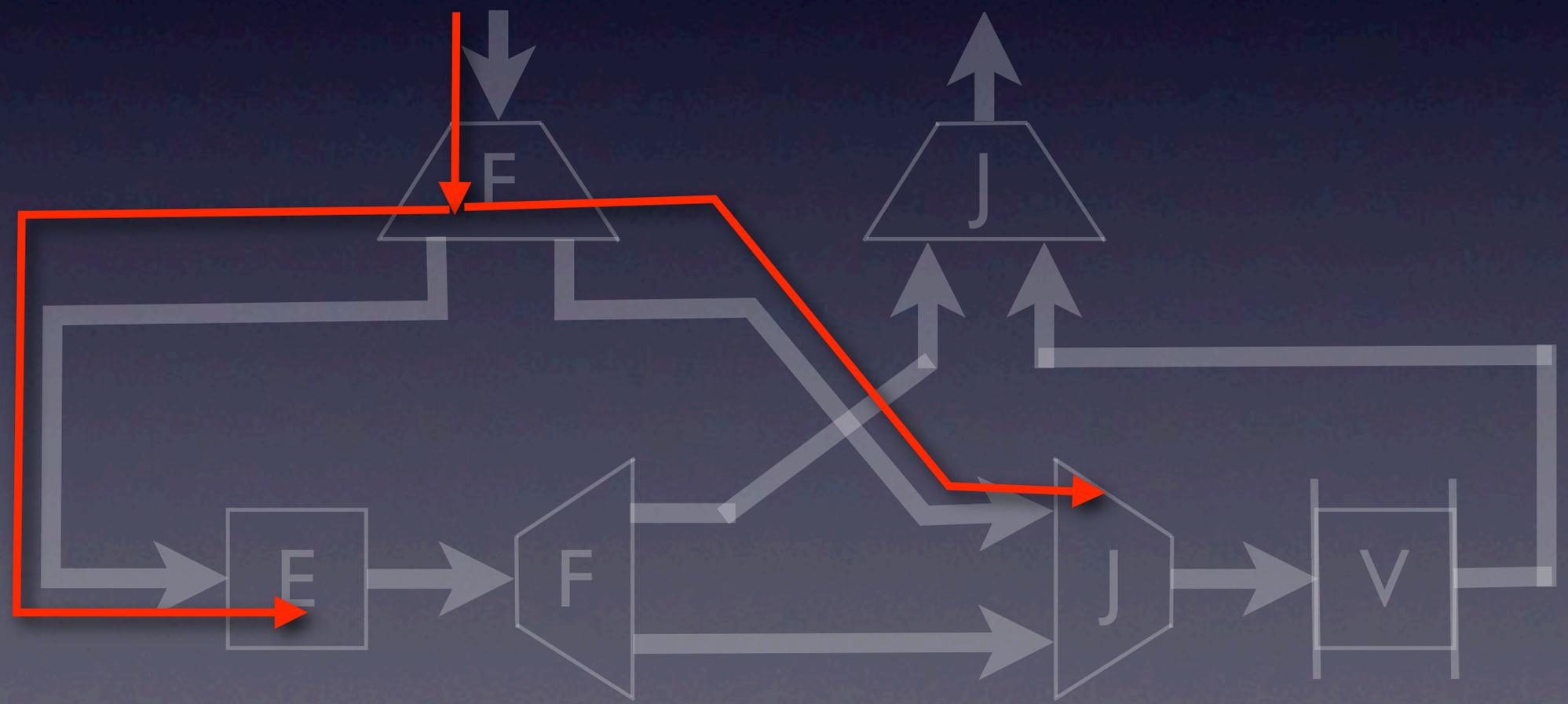
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



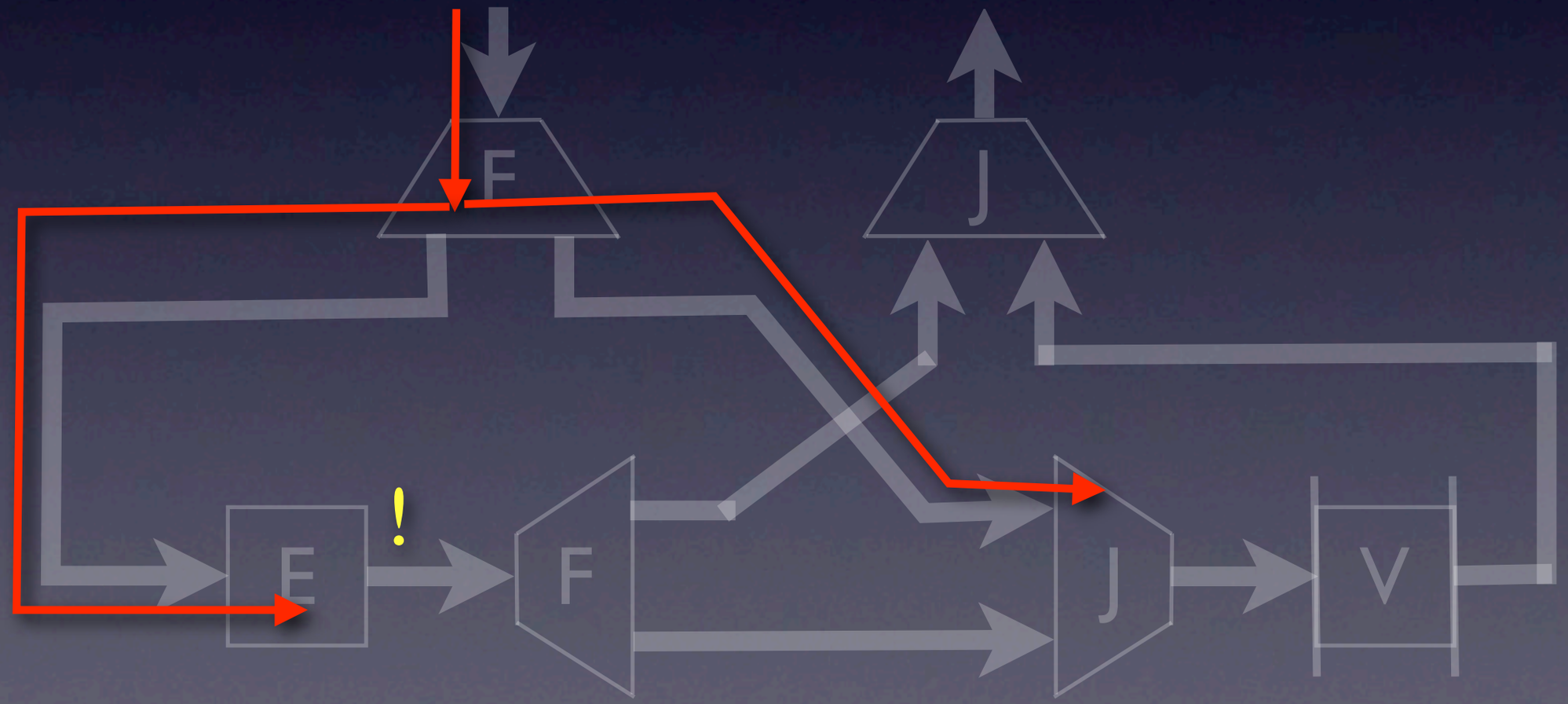
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



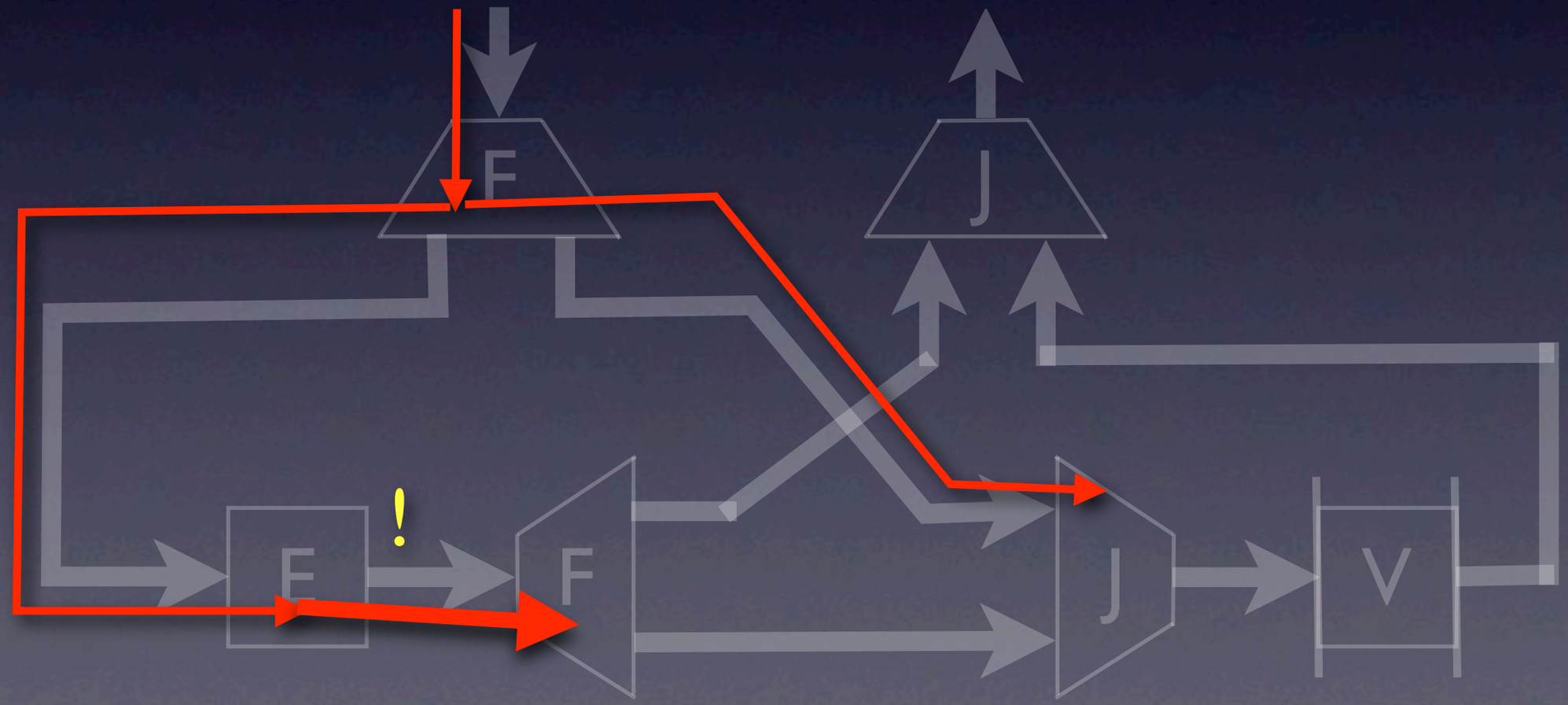
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



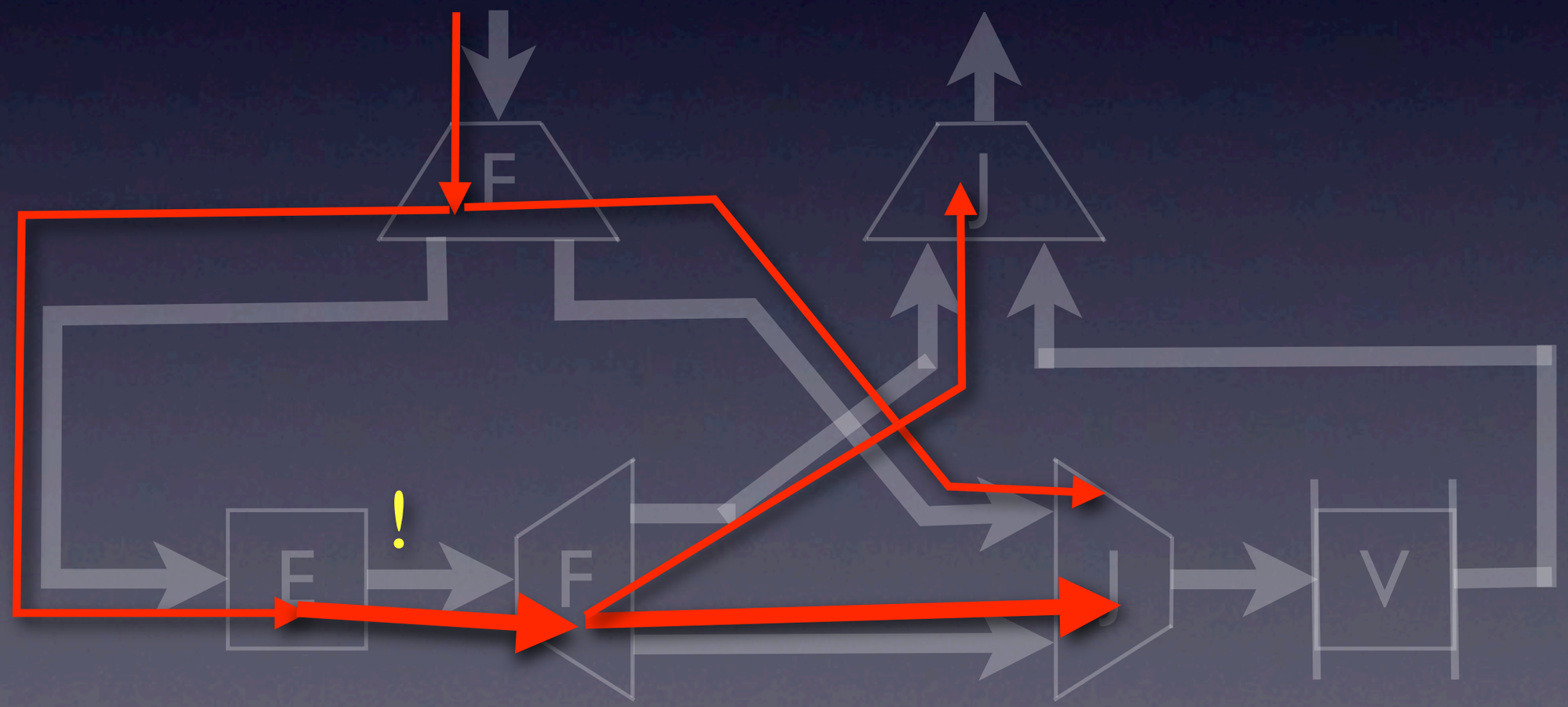
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



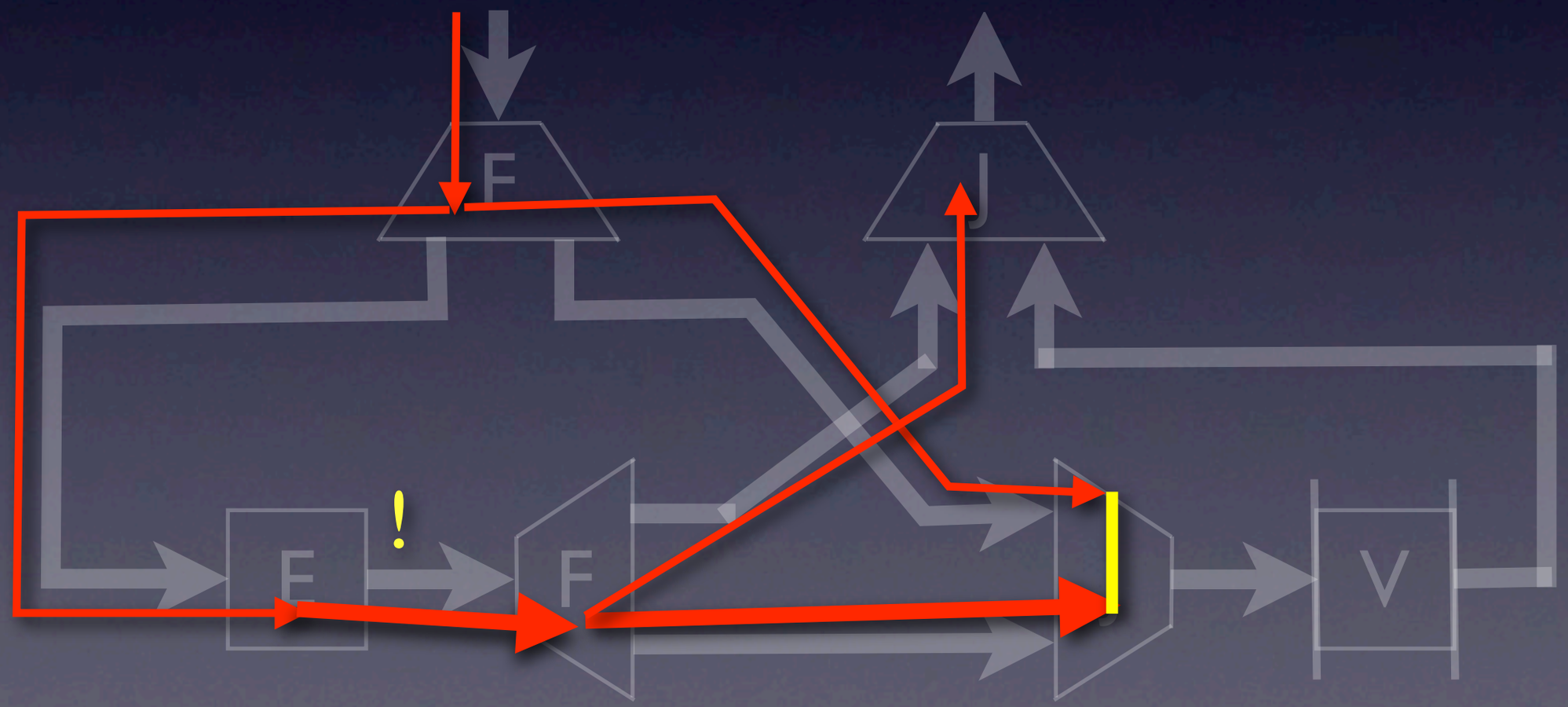
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



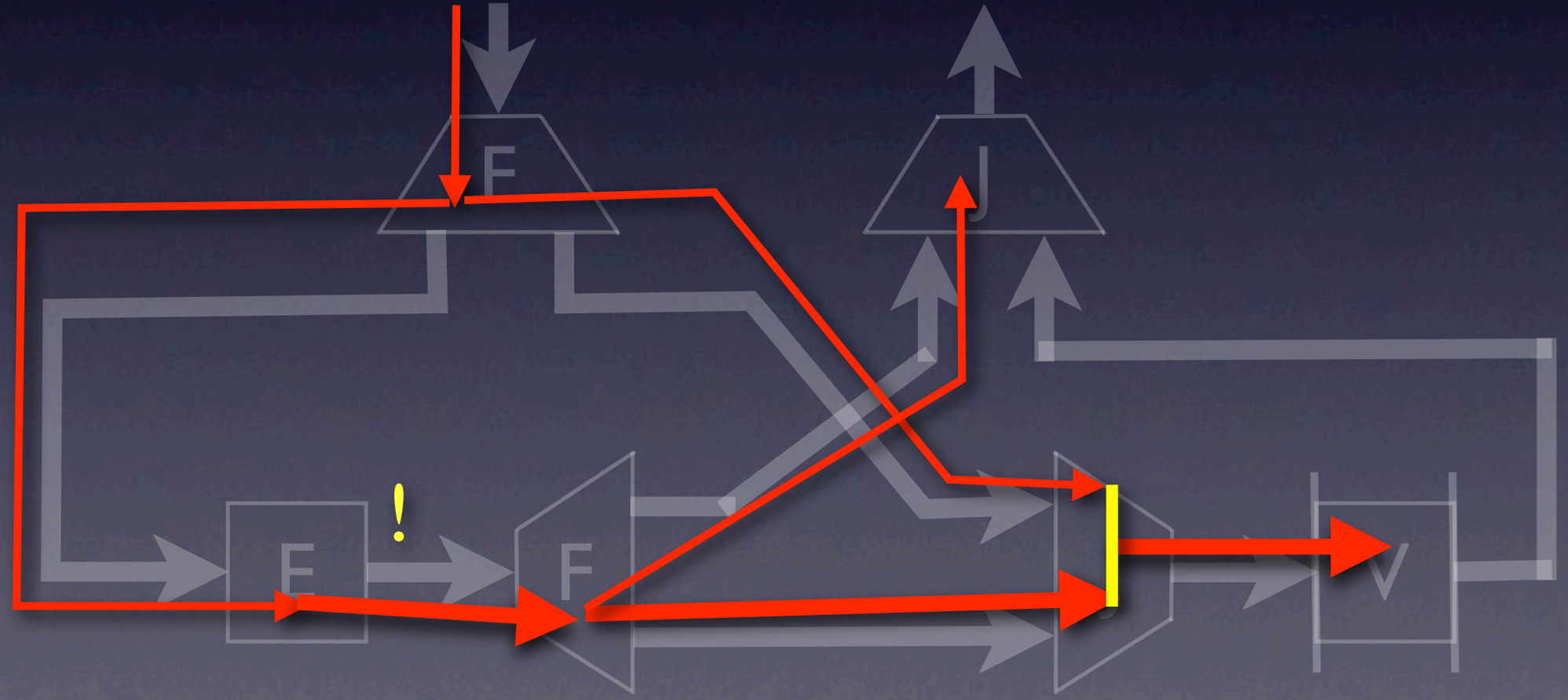
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



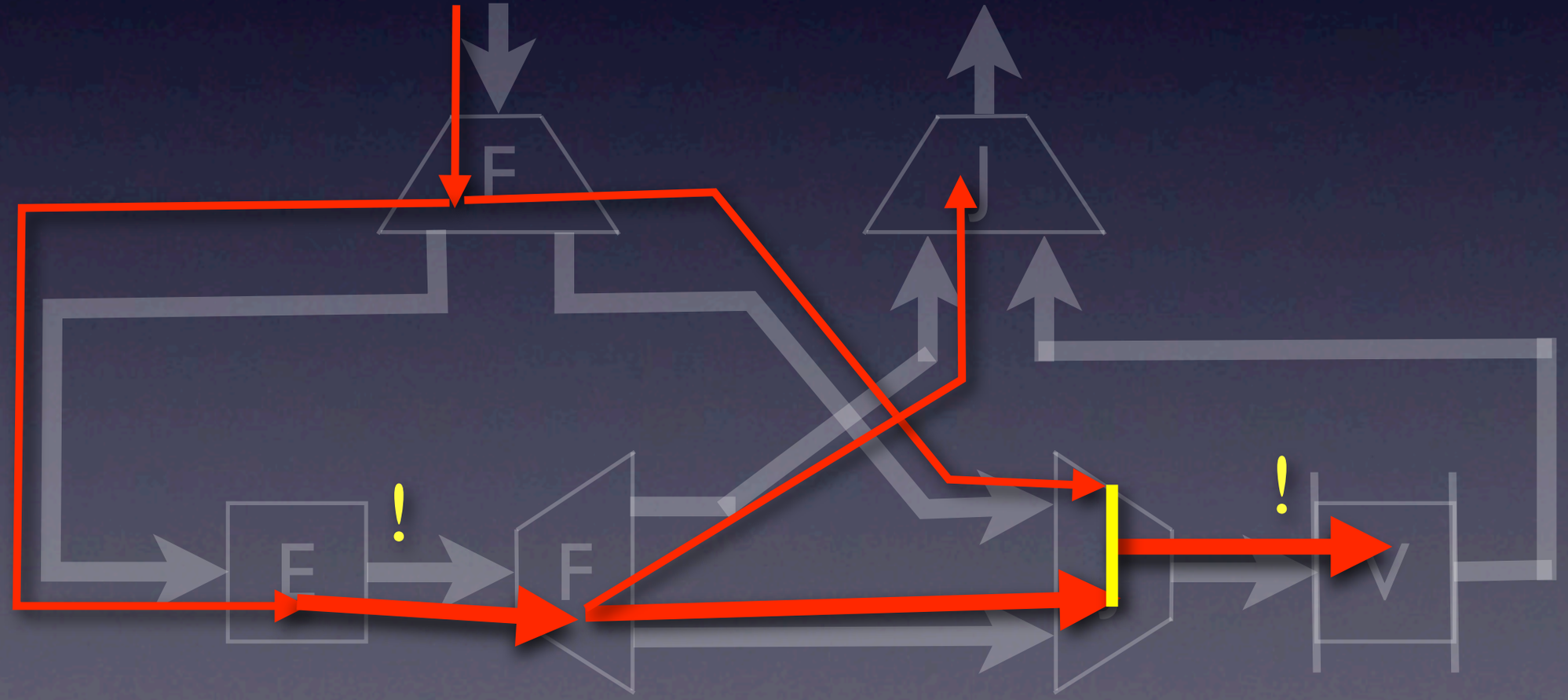
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



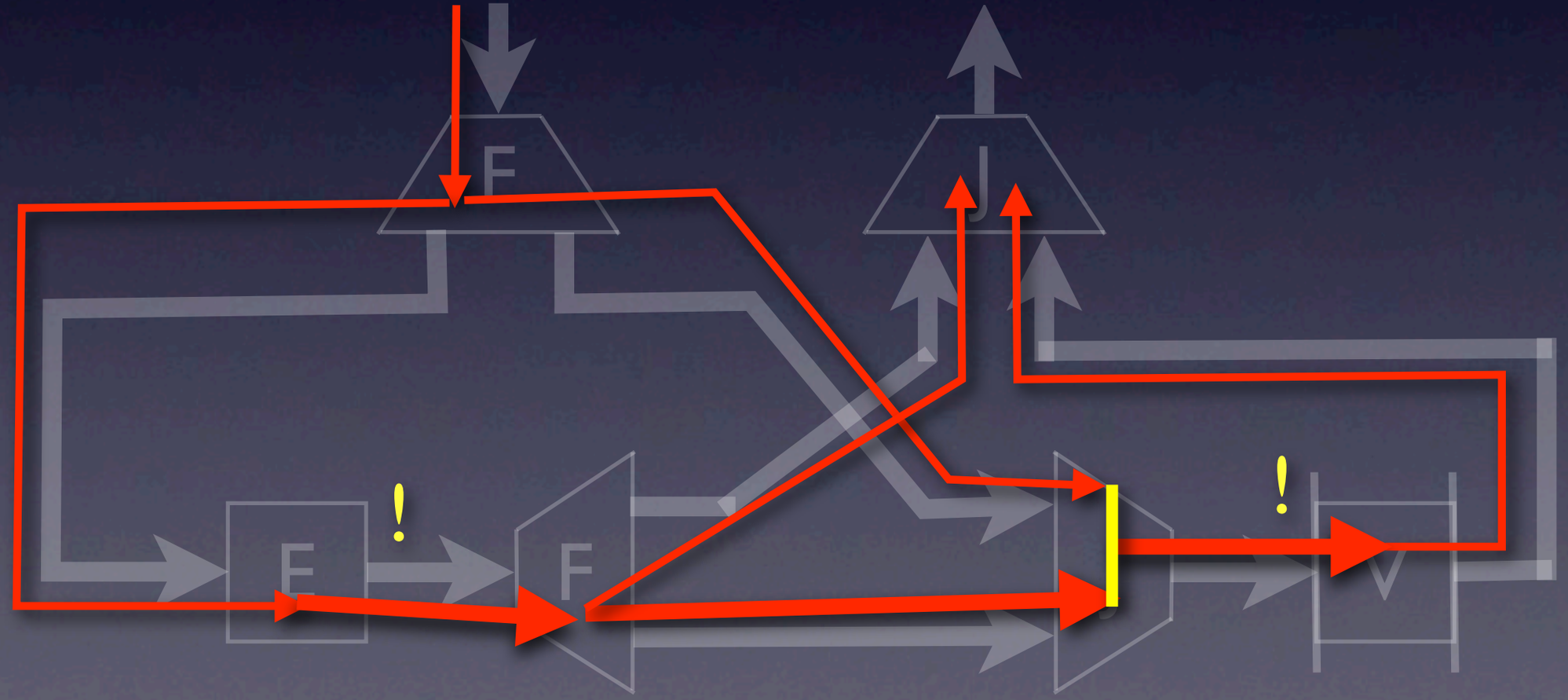
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



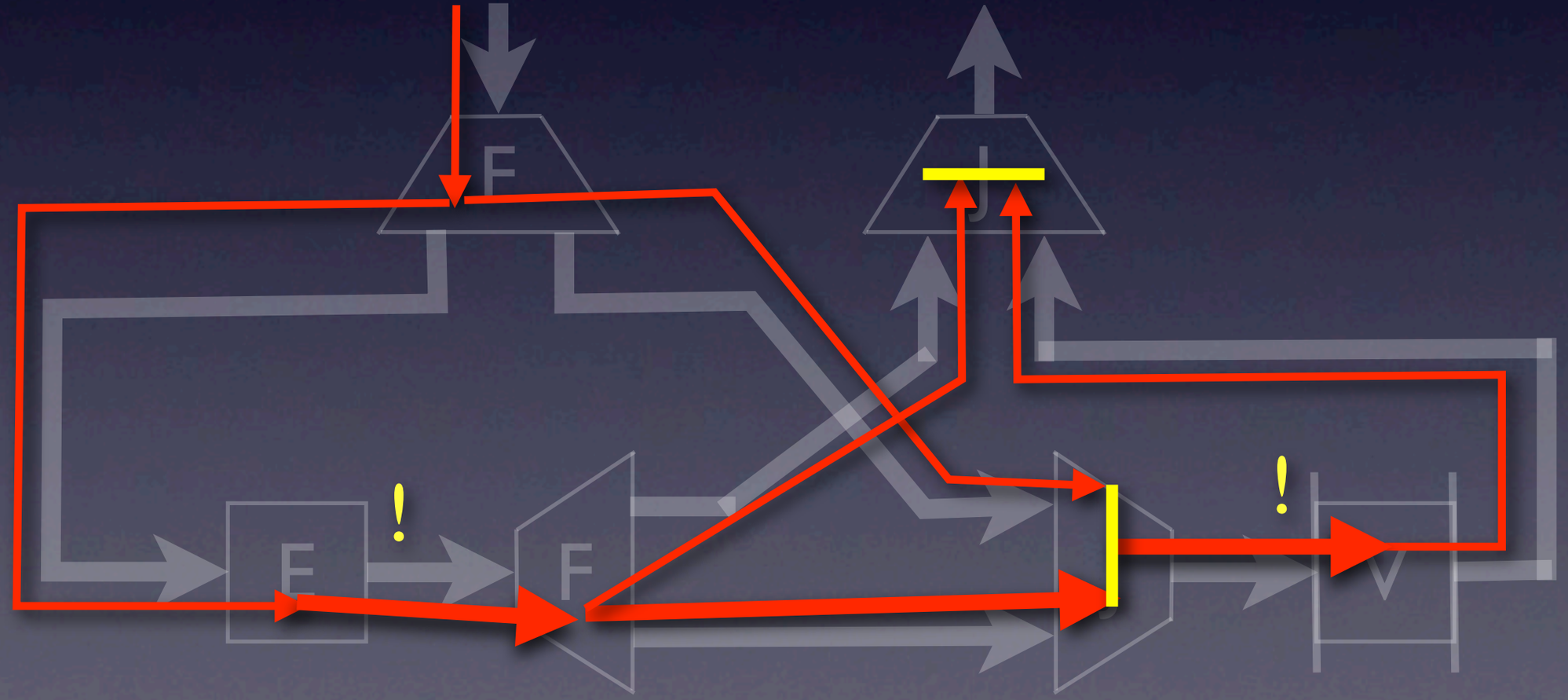
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



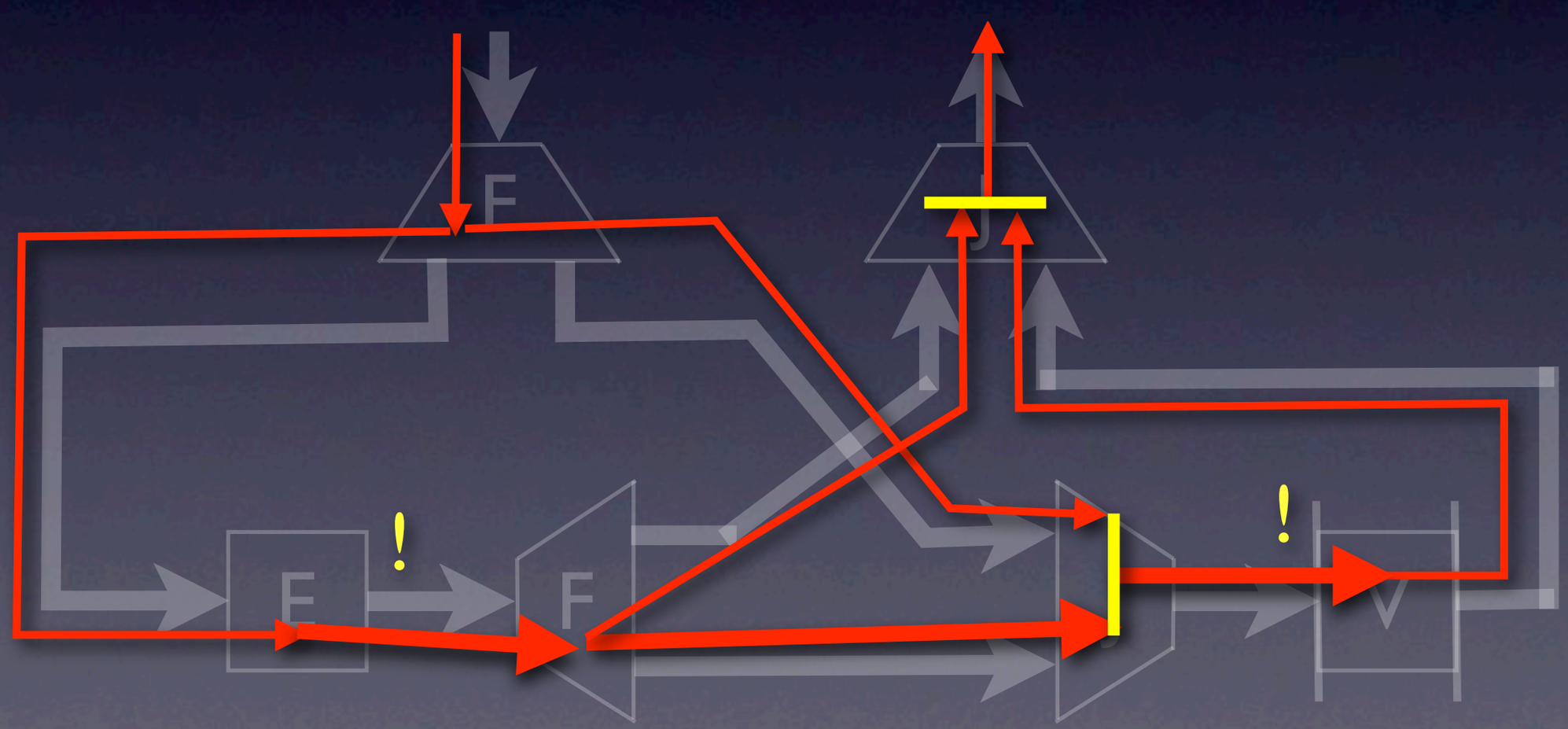
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



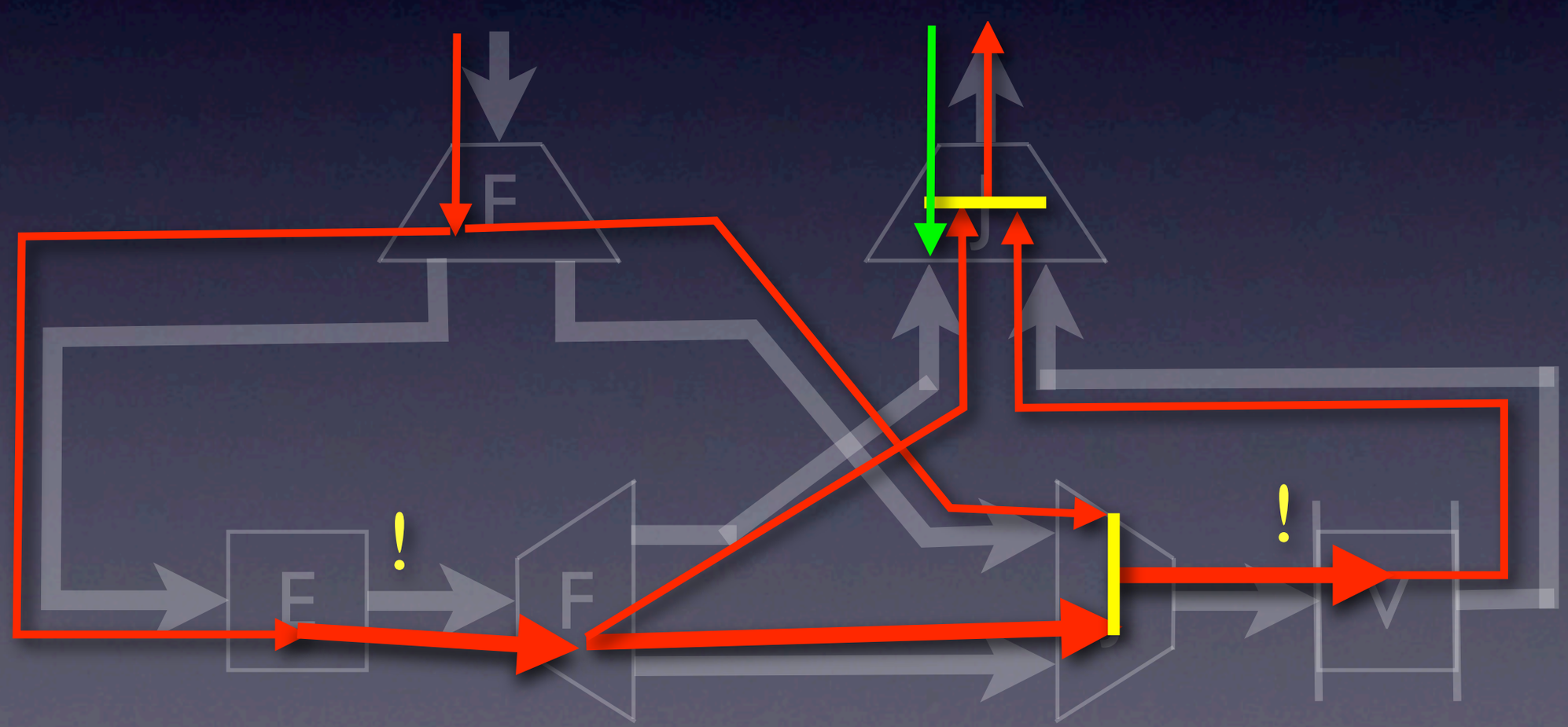
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



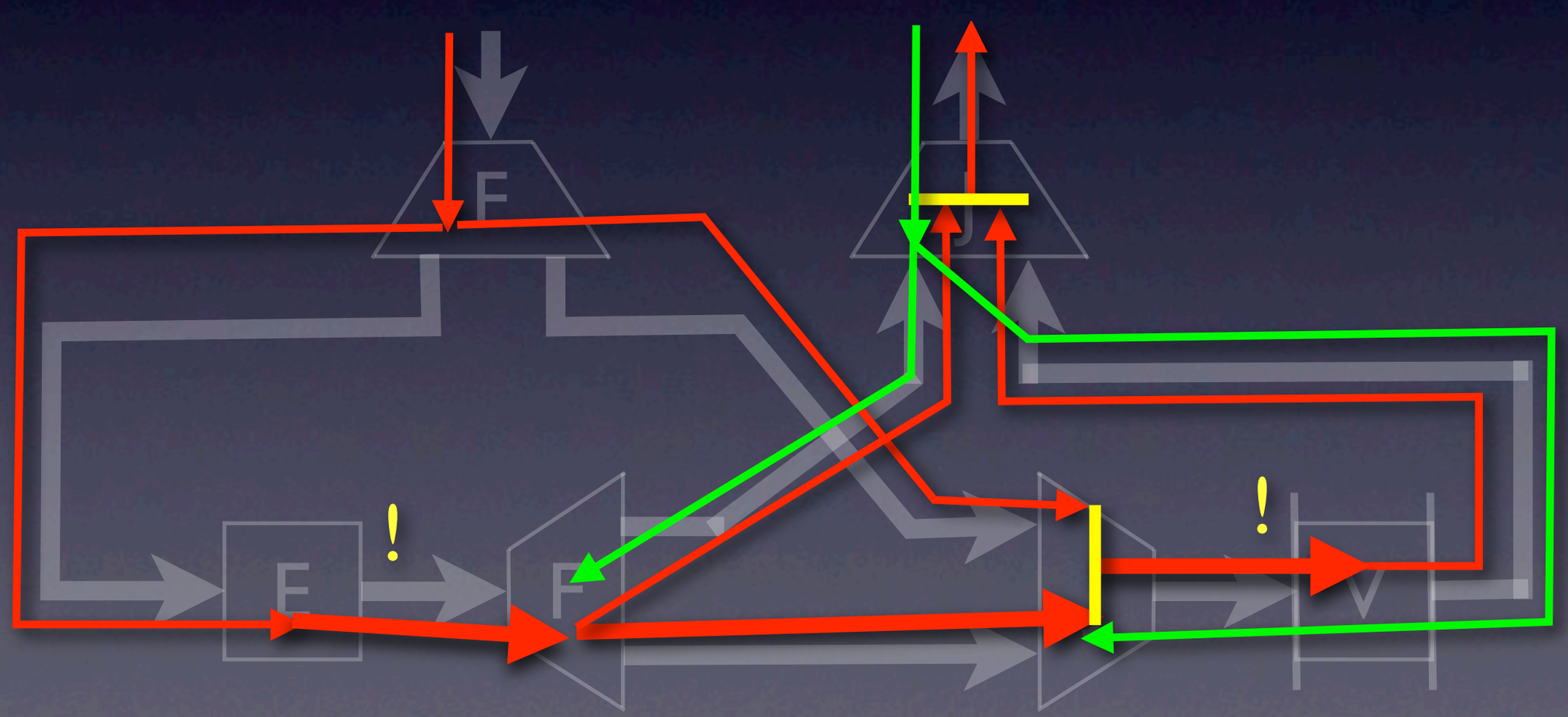
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



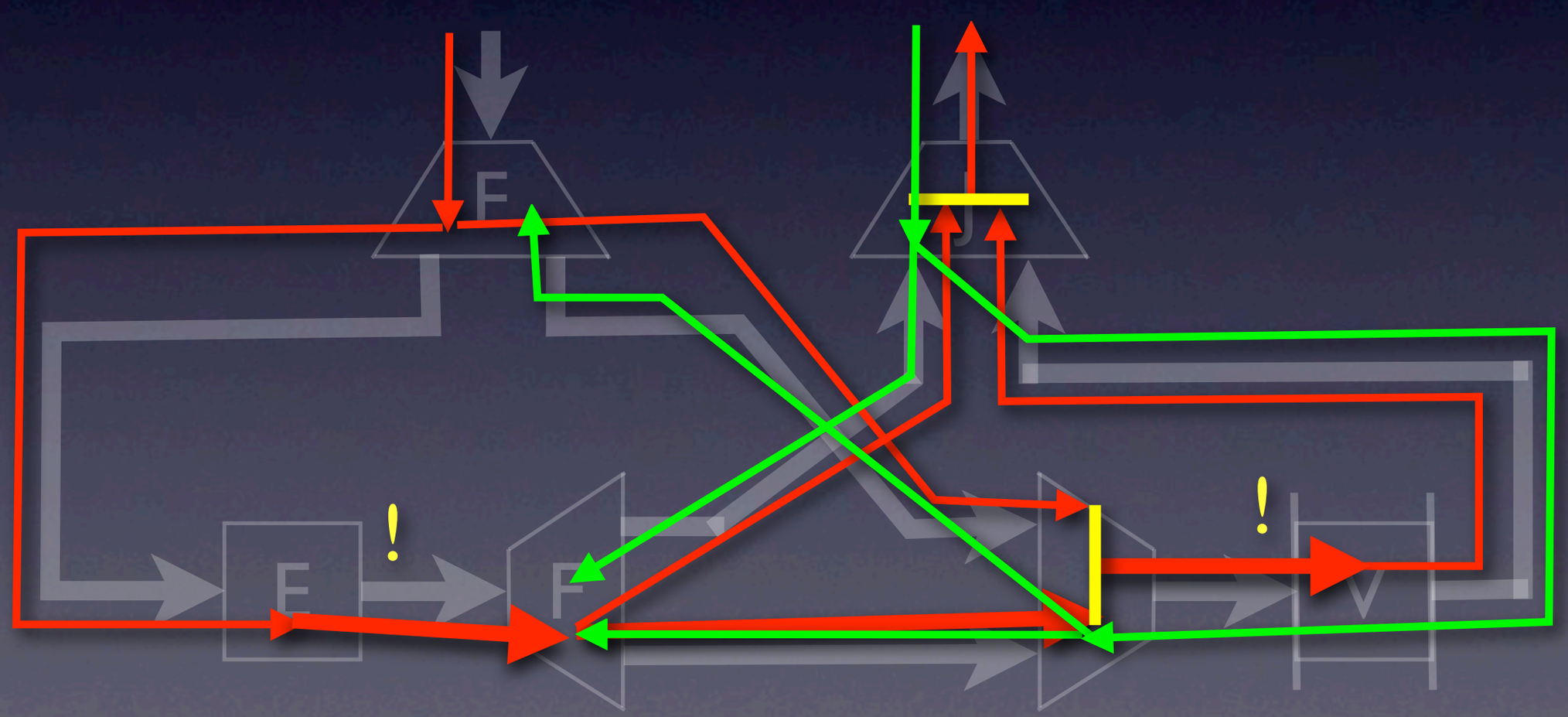
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



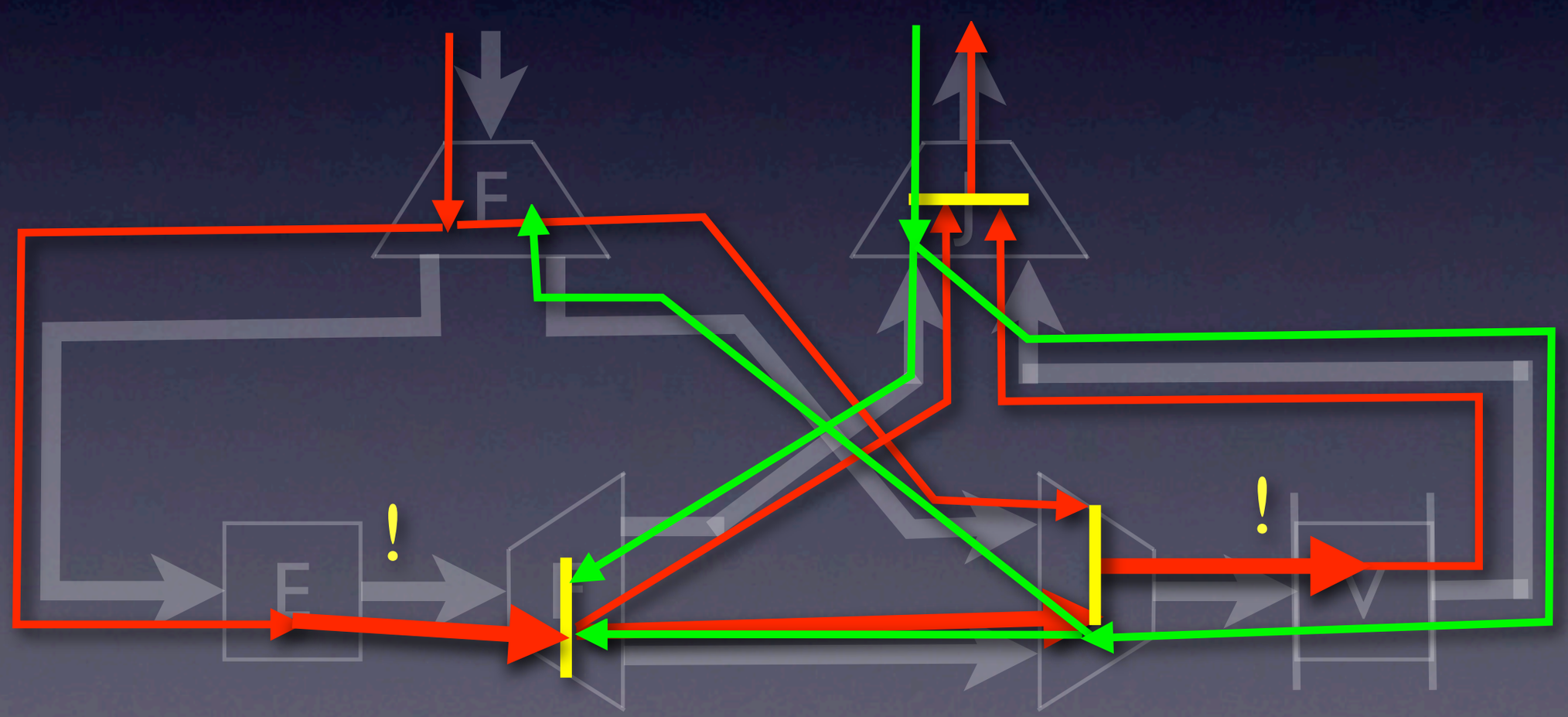
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



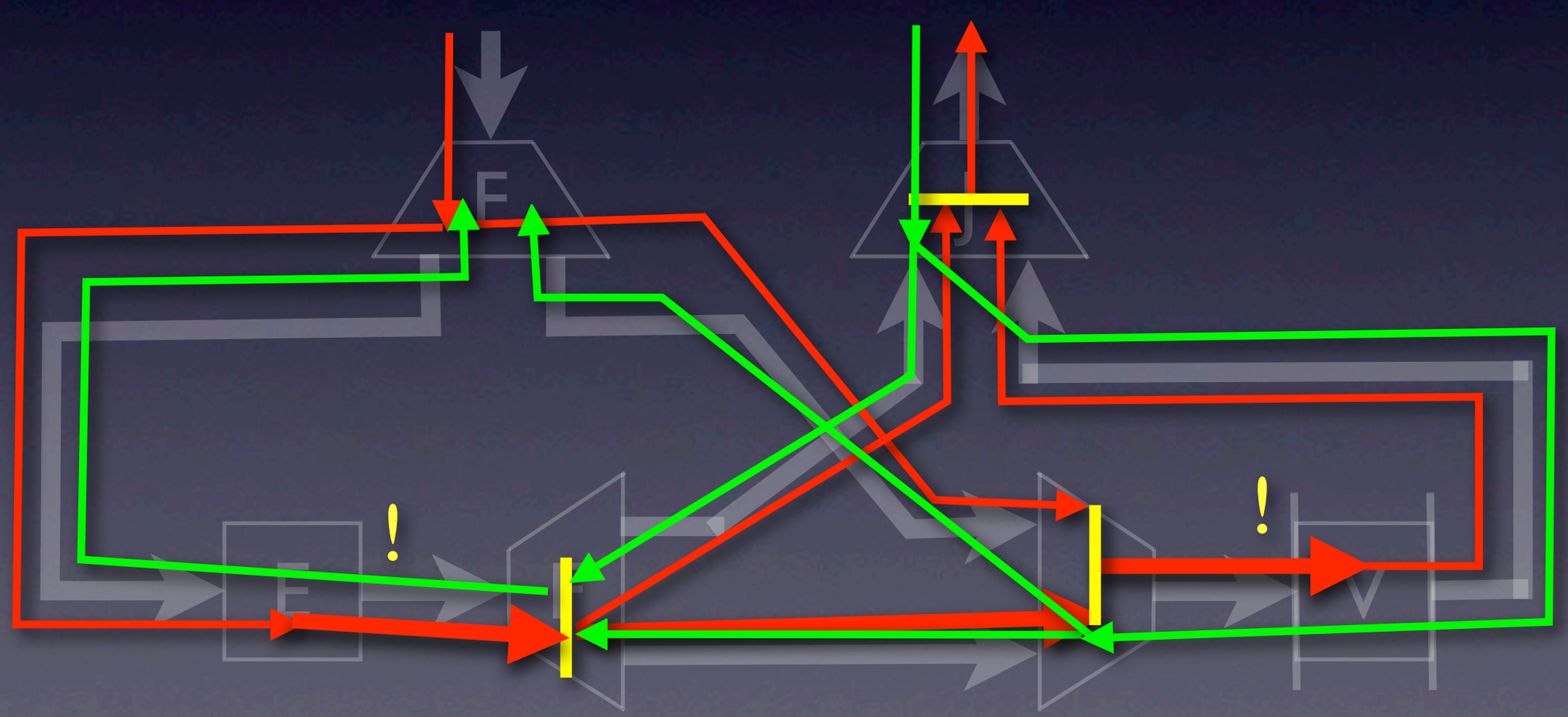
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



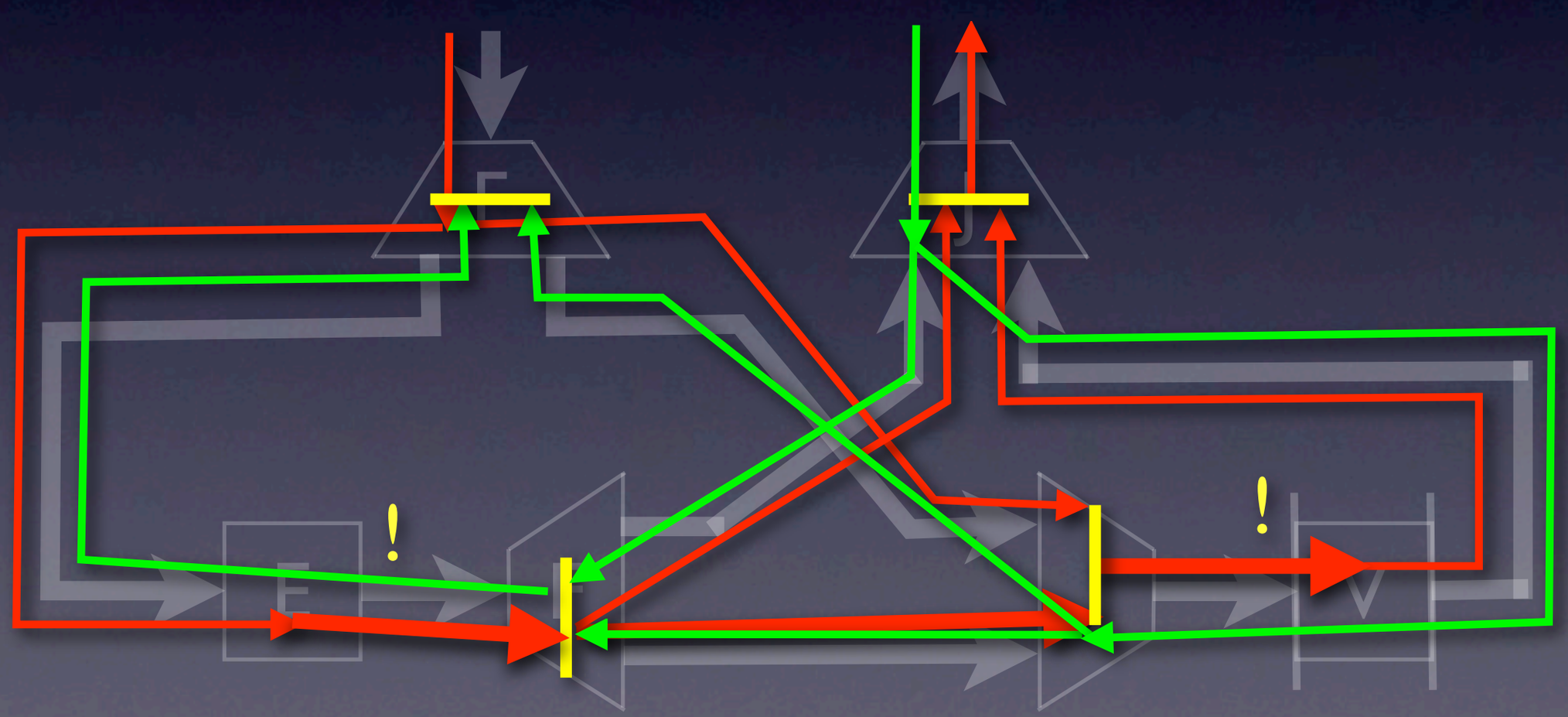
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



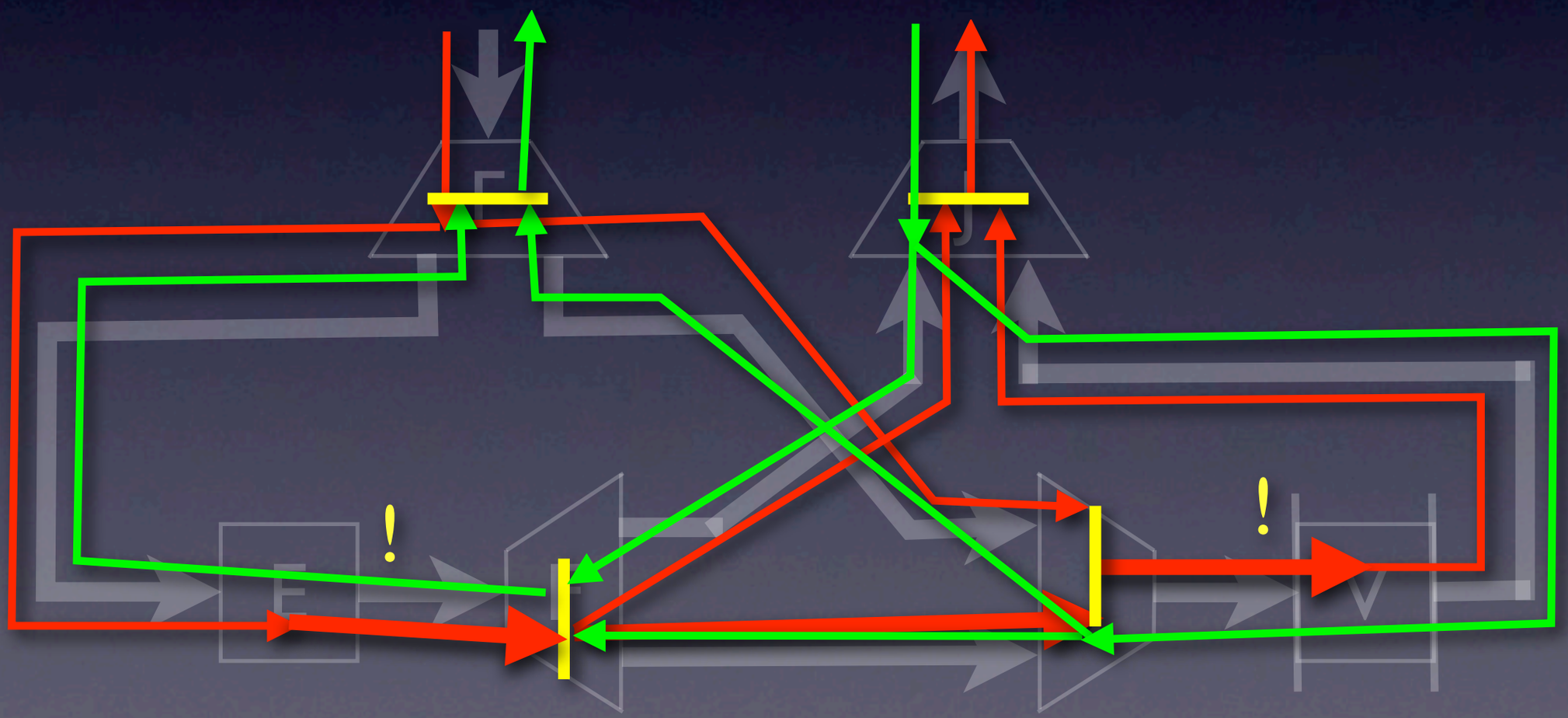
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



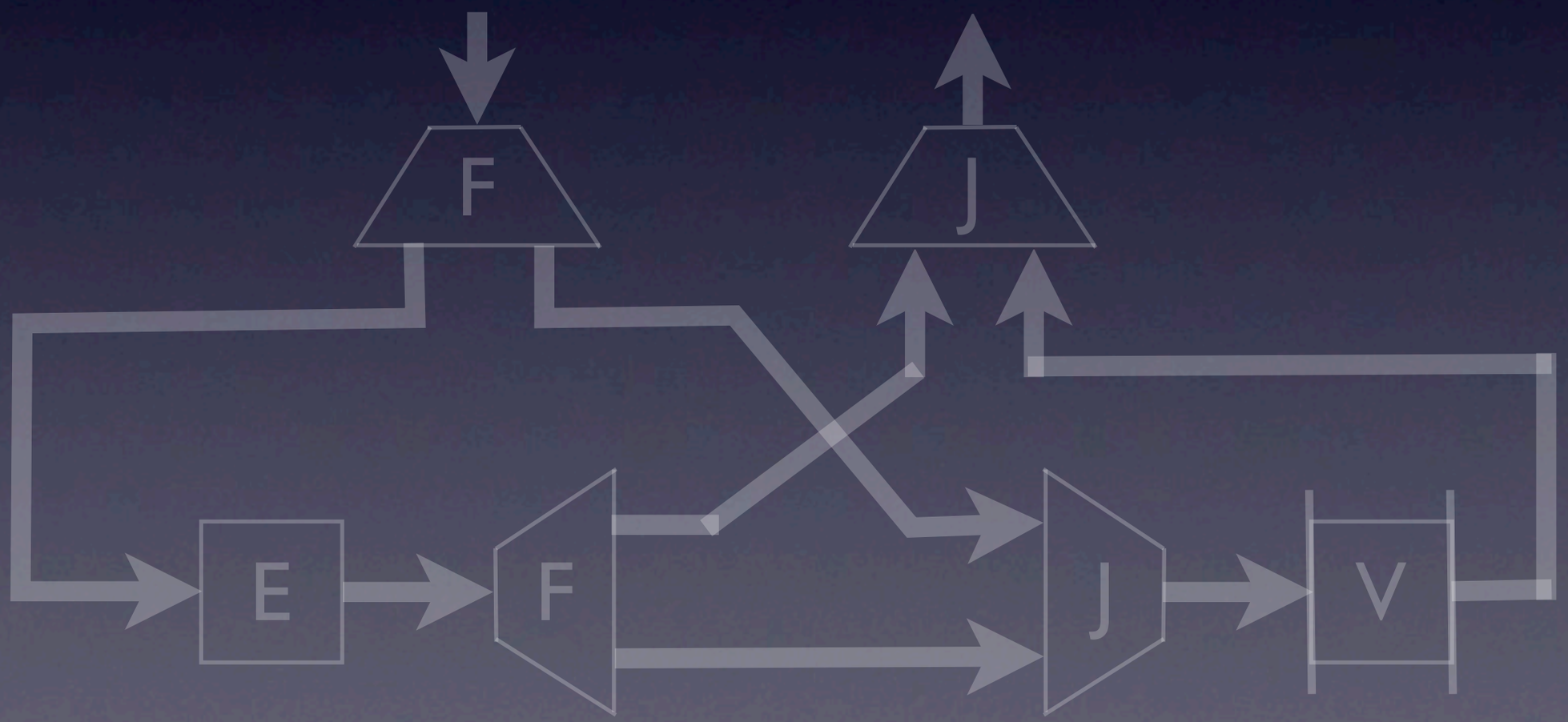
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



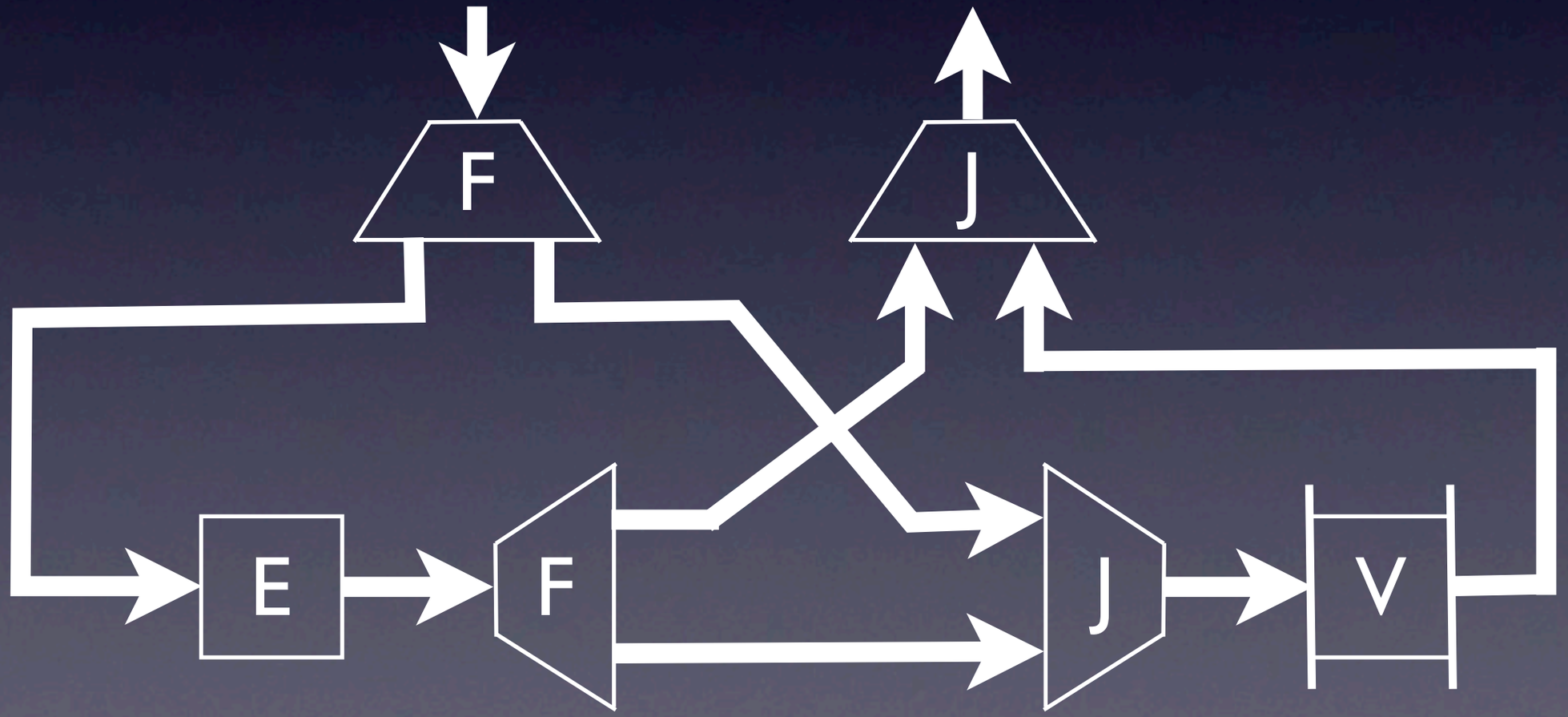
Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



Teak compilation

- Channel construction: `channel c : T`
  - `c <- E || c -> V`



The cost

- More components in unoptimised form
- Need to insert storage/handshake decoupling as a post-processing step
- The Balsa language no longer has as much guaranteed enclosure/sequencing
  - `select` not as useful
  - ‘stand-alone’ multiplexing more difficult

Tool setup

- New Balsa compiler: `teak`
  - Balsa -> Breeze. Targets:
    - `balsac` - balsa-c style HCs
    - `teak` - teak components
- balsa-netlist used for netlisting
  - teak components described in ABS

Completed

- `teak` compiler, ABS component descs.
- Sparkler - simple Sparc description
  - Simulated at gate level from teak
  - 200% gate count, 30% slower than example/dual_b. Not bad for a first cut
- nanoSpa - compiled, almost works

Still to do

- Pipeline latch insertion
  - currently inserting them everywhere
- Many peephole optimisations
- Get nanoSpa working, other examples
- Better component descriptions
- Behavioural simulation and visualisation