Todays schedule

- Asynchronous processing & tool-chain approach
- Integrity, privilege separation and capabilities.
- CarvFS & MinorFS
- MattockFS core design
- **MattockFS as distributed-framework building block**
- Installation (hands on)
- File-system as API (hands on)
- Python API (hands on)

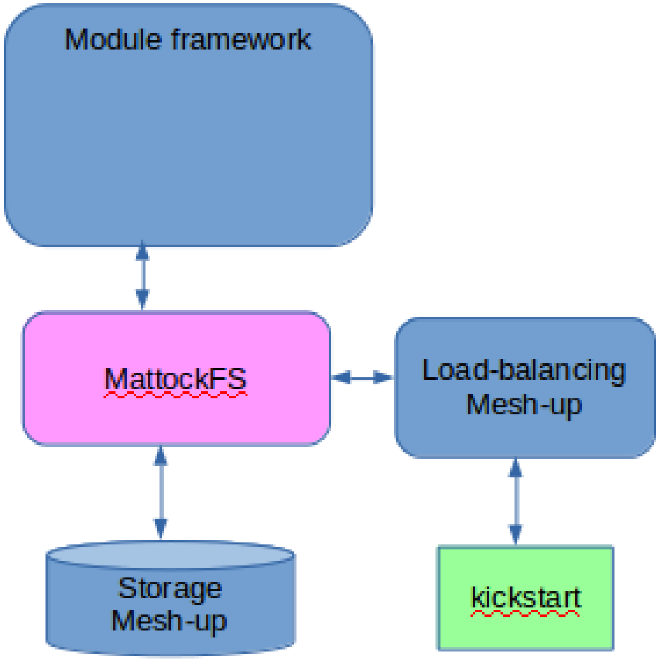
MattockFS

Computer-Forensics File-System

MattockFS as distributed-framework building block.

High level view

- MattockFS as building block
- Module framework
  - Possibly purely a library
- Load-balancing mesh-up
- Remote kick-start
- Storage mesh-up



Base facility for storage mesh-up

- /var/mattock/mnt/$MPNO
  - Put mnt/0 on local storage
  - Put mnt/1 .. mnt/n on NFS shares MattockFS nodes

Considerations for kickstarting

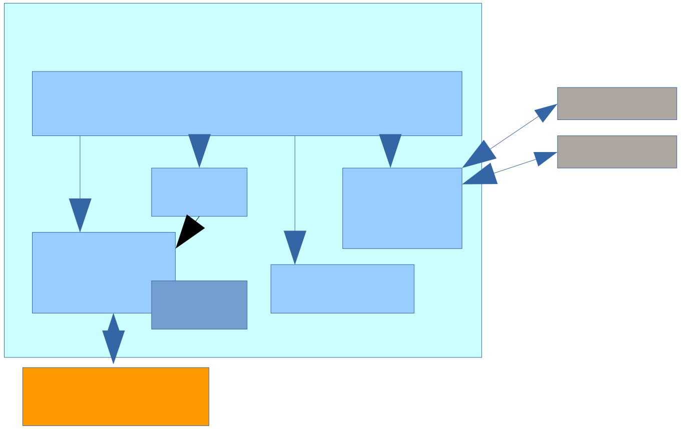
- Networking client for kickstarting to MattockFS node.
- “Most” of the data from a kick-started image should remain on initially chones MattockFS node! (locality of data concerns).
- If throttling on the client(s) isn’t an option, consider providing a proxied kick-start.

Load-balancing mesh-up

- MattockFS has hooks for ‘stealing’ most CPU intensive jobs from actors for load balancing purposes.
- In case of unbalance CPU usage, load-balancer ends tool-chain on node0.
- Router state of prematurely completed tool-chain.
- New tool-chain initiated on nodeX using old router state.

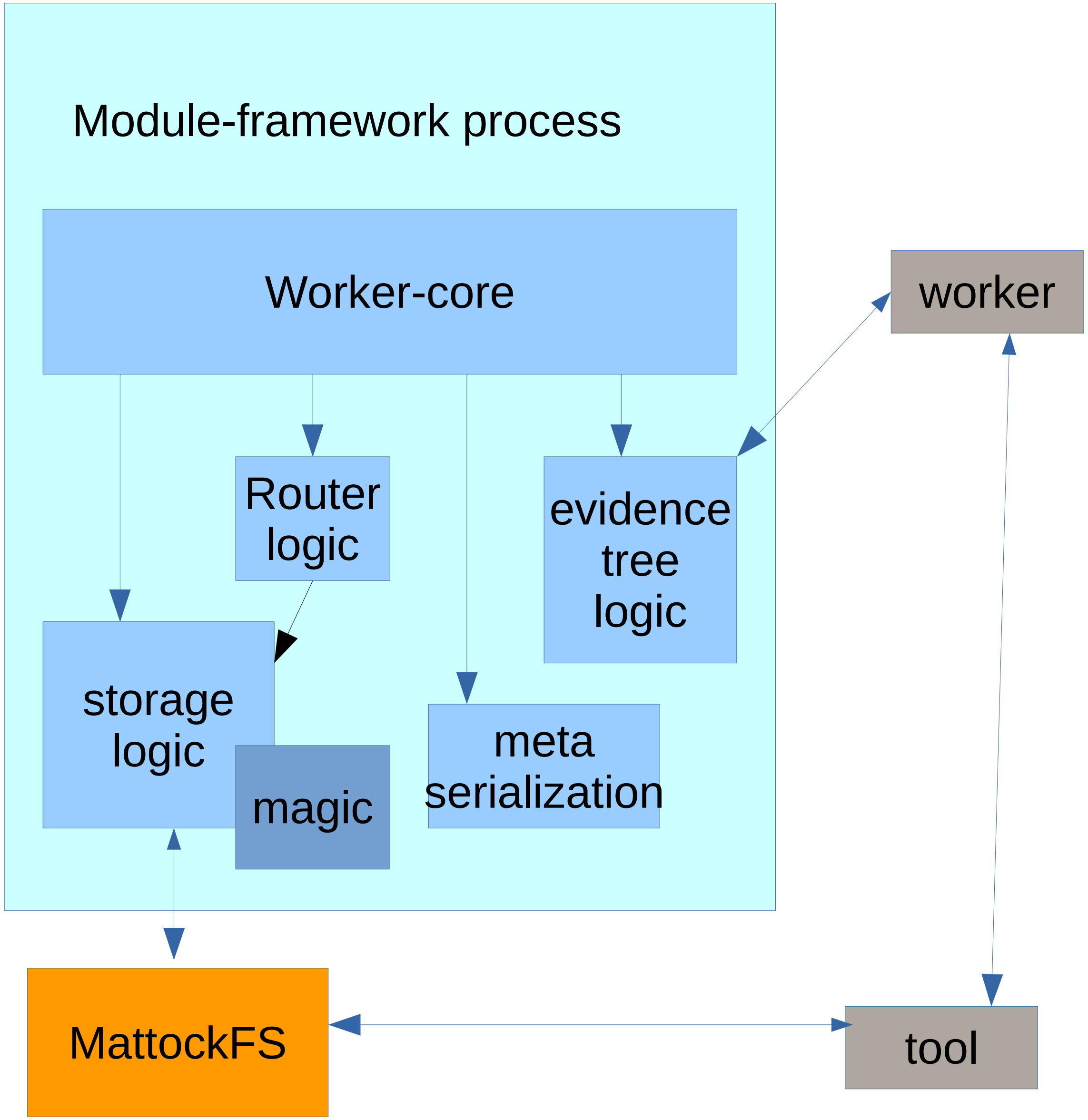
Module framework library

- Built on top of low level MattockFS language binding and CarvPath library
- Need a module framework for each language that needs to be supported!



Module framework library

- Magic library
- Meta serialization
- Distributed router logic.
- Evidence module API
- Storage logic with throttling support.



Magic library

- In OCFA most tool-chains included file module
- Many tool-chains completed after file-type check.
- Integrating libmagic in each module reduces unneeded IPC.
- Requirement: module framework library MUST include libmagic functionality.

Meta Serialization

- OCFA used XML and relied heavily on XSLT
  - Serious concerns high-volume processing performance.
- Suggestion: Serialization technology should be chosen carefully.

Distributed router logic

- OCFA used a central XML router
  - This doubled messaging
- Original OCFA router was stateless.
  - Modeled after IPTABLES
- The FIVES project introduced router-state (line number) to tool-chain.
- MattockFS API includes limited space for tool-chain level router state.
- Requirement: module framework library MUST provide distributed routing functionality.
- Suggestion : module framework library should explore a rich yet minimally statefull routing logic language.

Module API

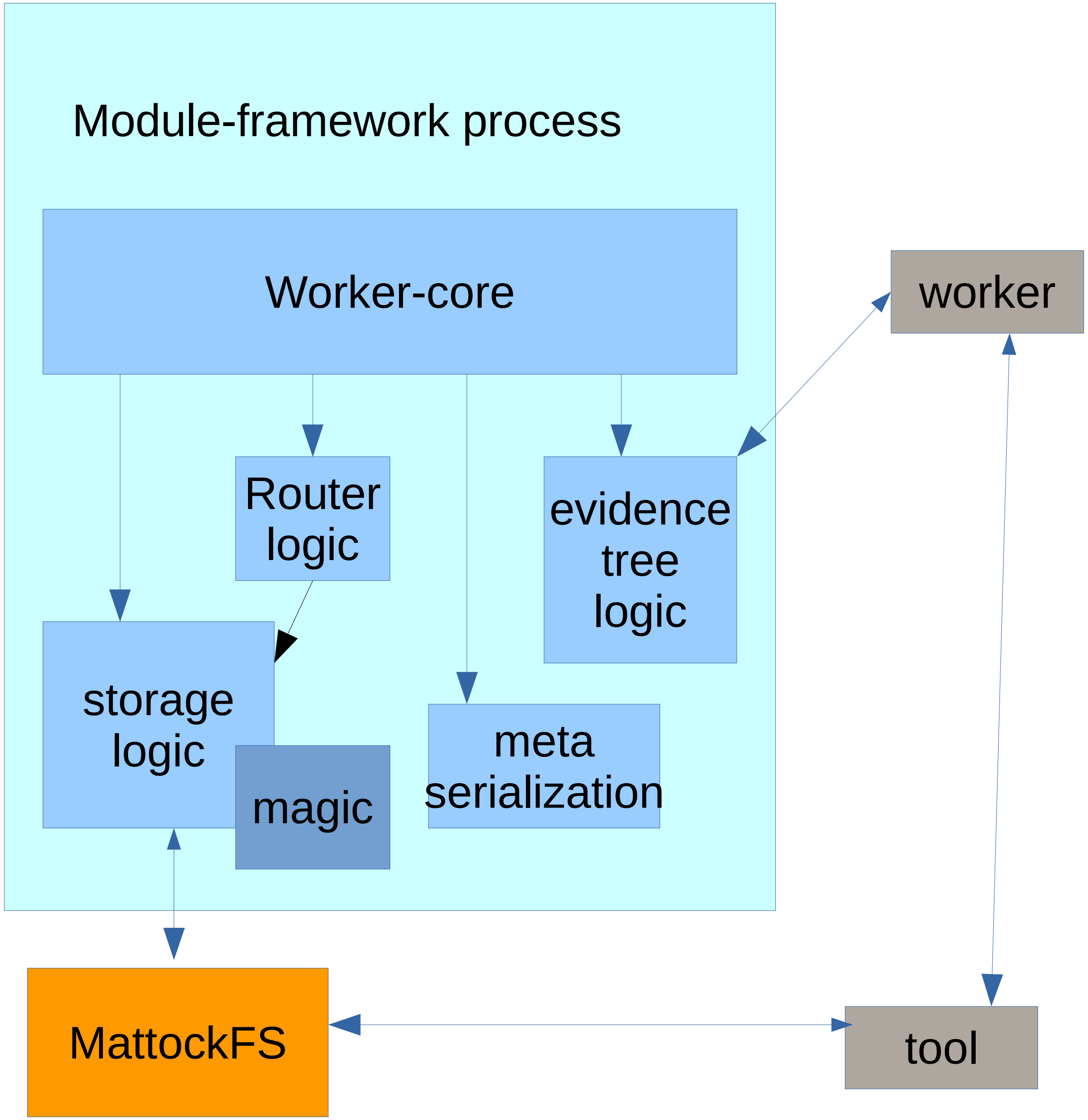
- OCFA started off with a simple single-callback API (processEvidence).
- Later versions of OCFA also included a tree-graph API.
- The tree-graph API turned out to be much more powerful w.r.t. deep meta-data.
- Suggestion: new API should preferably use a tree-graph API.

Throttling

- MattockFS provides hooks for throttling related info.
- The OS provides additional information needed for throttling purposes.
- Requirement: The module framework library MUST implement throttling.

Module framework library

- Magic library
- Meta serialization
- Distributed router logic.
- Evidence module API
- Storage logic with throttling support.


Module-framework process
Worker-core
Router logic
evidence tree logic
storage logic
magic
meta serialization
worker
MattockFS
tool