

Inside Broker

How Broker Leverages the C++ Actor Framework (CAF)

Dominik Charousset

iNET RG, Department of Computer Science
Hamburg University of Applied Sciences

Bro4Pros, February 2017

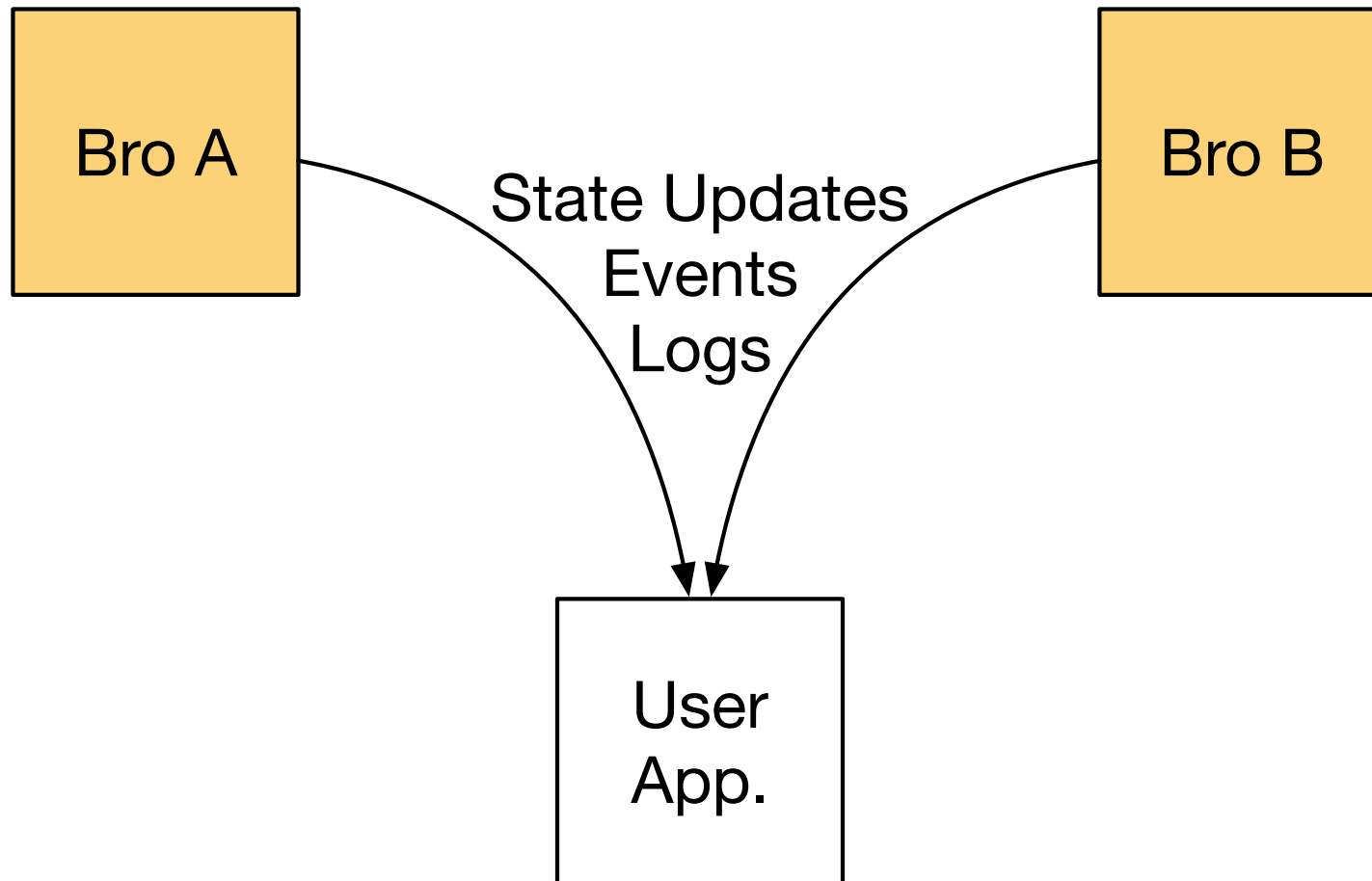


Hochschule für Angewandte
Wissenschaften Hamburg
Hamburg University of Applied Sciences



What was Broker
again?

Problem at Hand



Traditional Approach

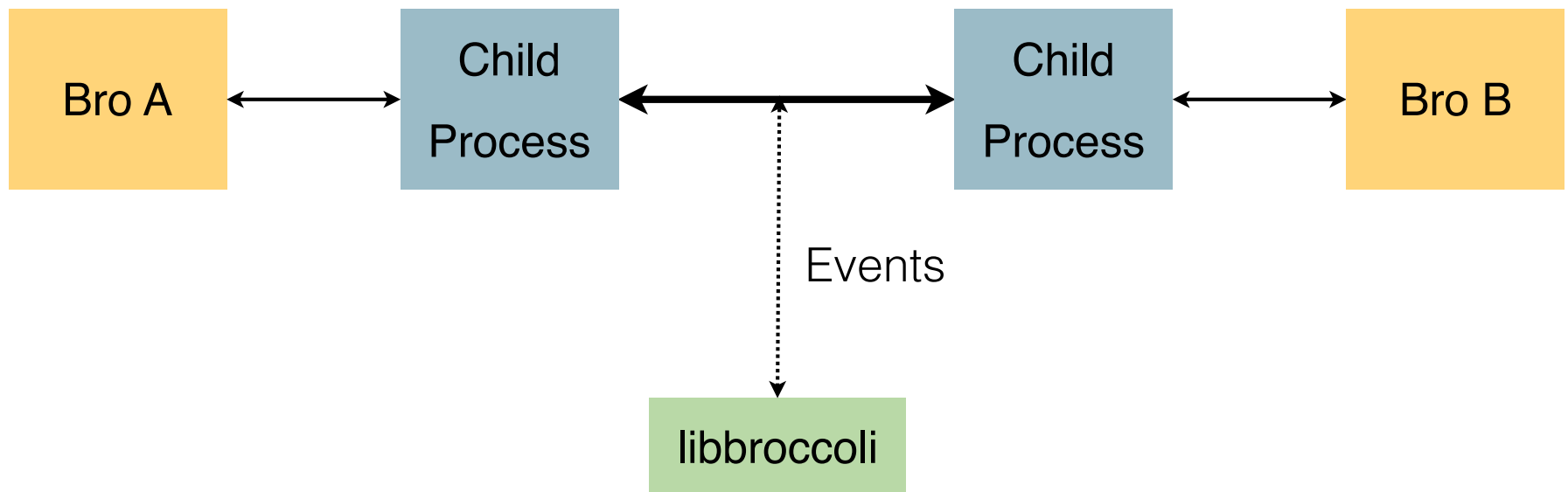


Image source: Robin Sommer, BroCon 2015

Traditional Issues

- Persistency issues
- Possible race conditions with &synchronized
- Limited control over data flow

Broker Approach

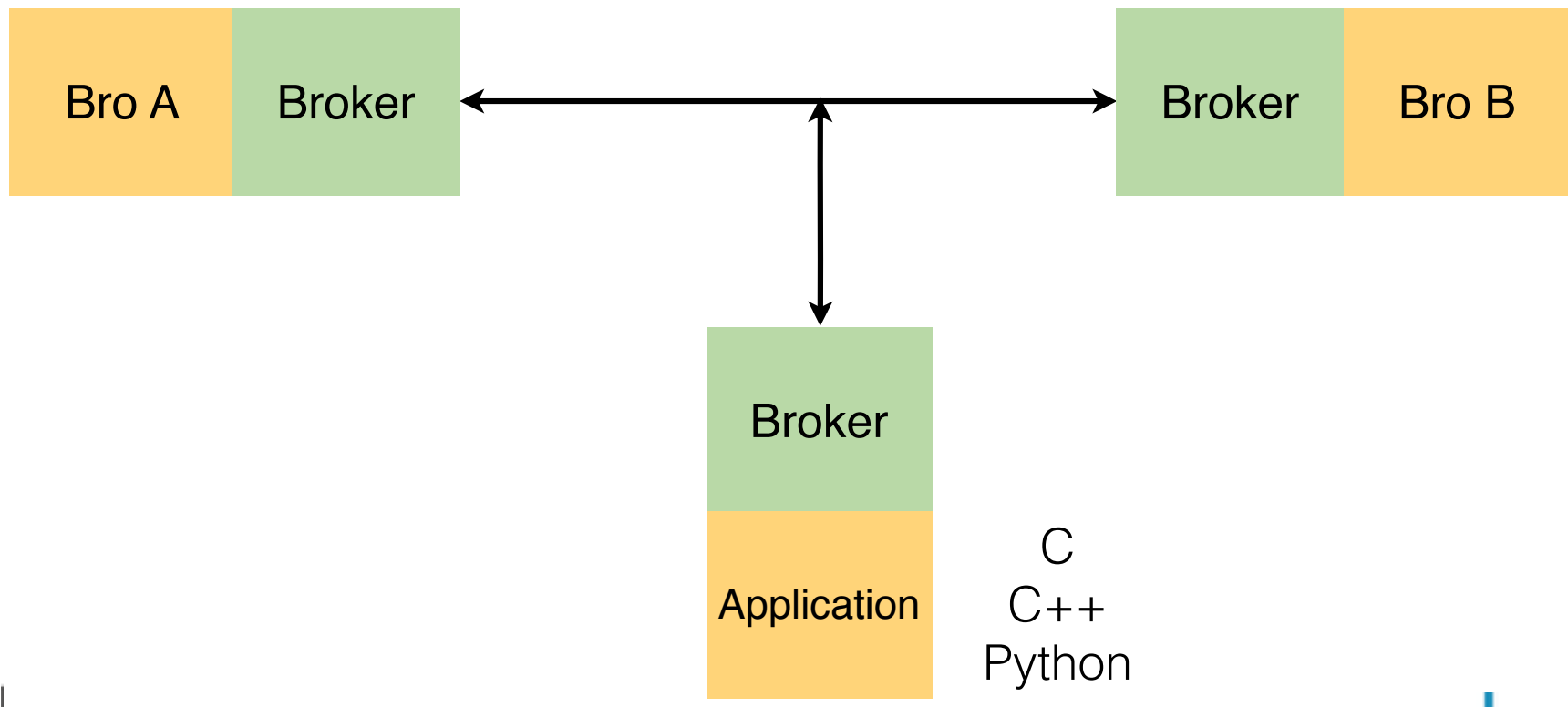


Image source: Robin Sommer, BroCon 2015

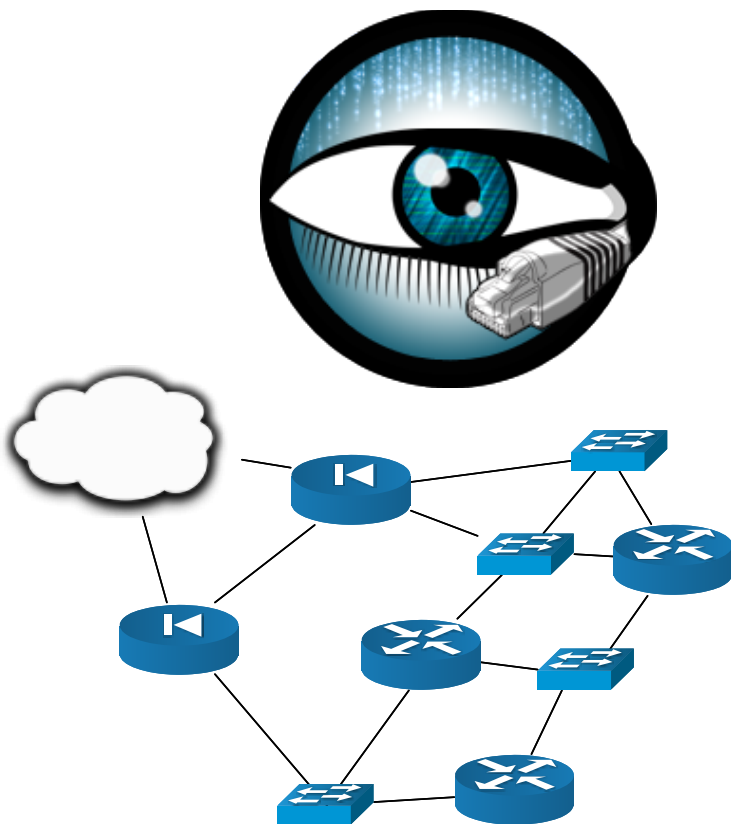
Broker Benefits

- Grant unified access to Bro events
- Empower users to manage state
- Provide a global, persistent key/value store

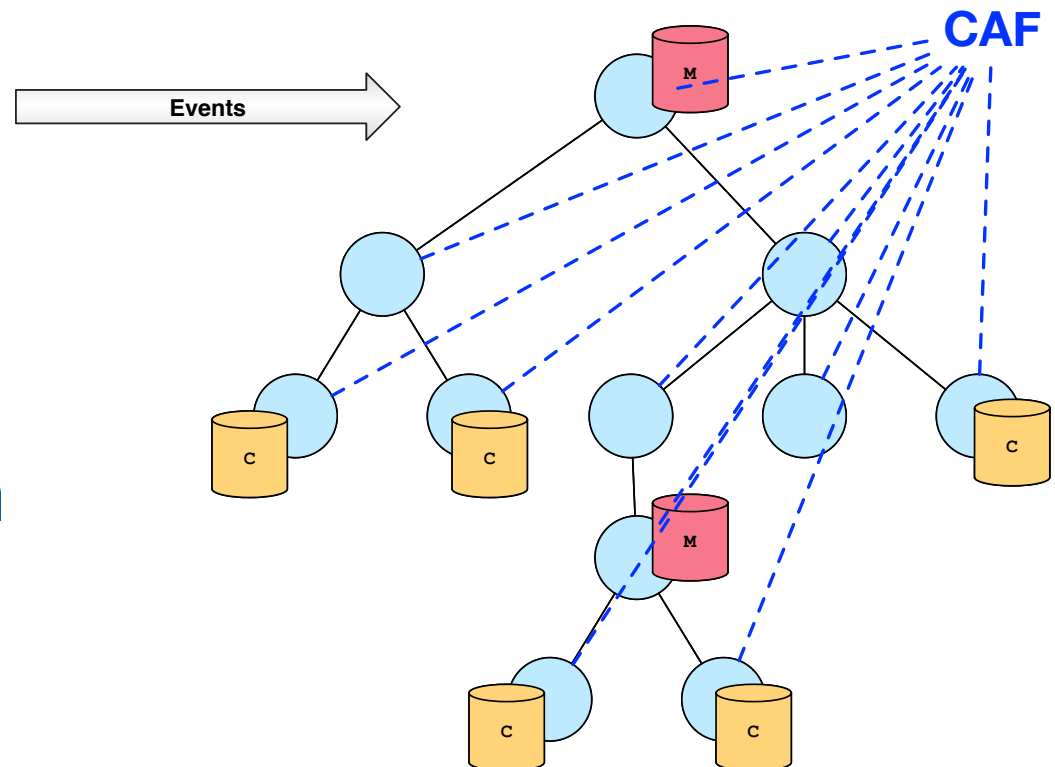
How does CAF relate
to Bro?

Broker in Context

Bro: monitor the network



Broker: distribute network insights



Broker's Goals

- Provide **flexible pub/sub** data distribution
- Enable **distributed**, deep detection
- Support **data-intense algorithms** on realtime events

Broker's Requirements

- Efficient communication layer
- Expressive data model
- Persistent storage

Fueling Broker

- Broker uses **CAF** to meet its requirements:
 - **Structure**: endpoints & messages
 - **Communication**: send & receive
 - **Network**: connect peers & distribute data

CAF in a Nutshell

- Programming interface based on the actor model
- Configurable runtime for infrastructure software*
- Emphasis on reliability, efficiency & maintainability

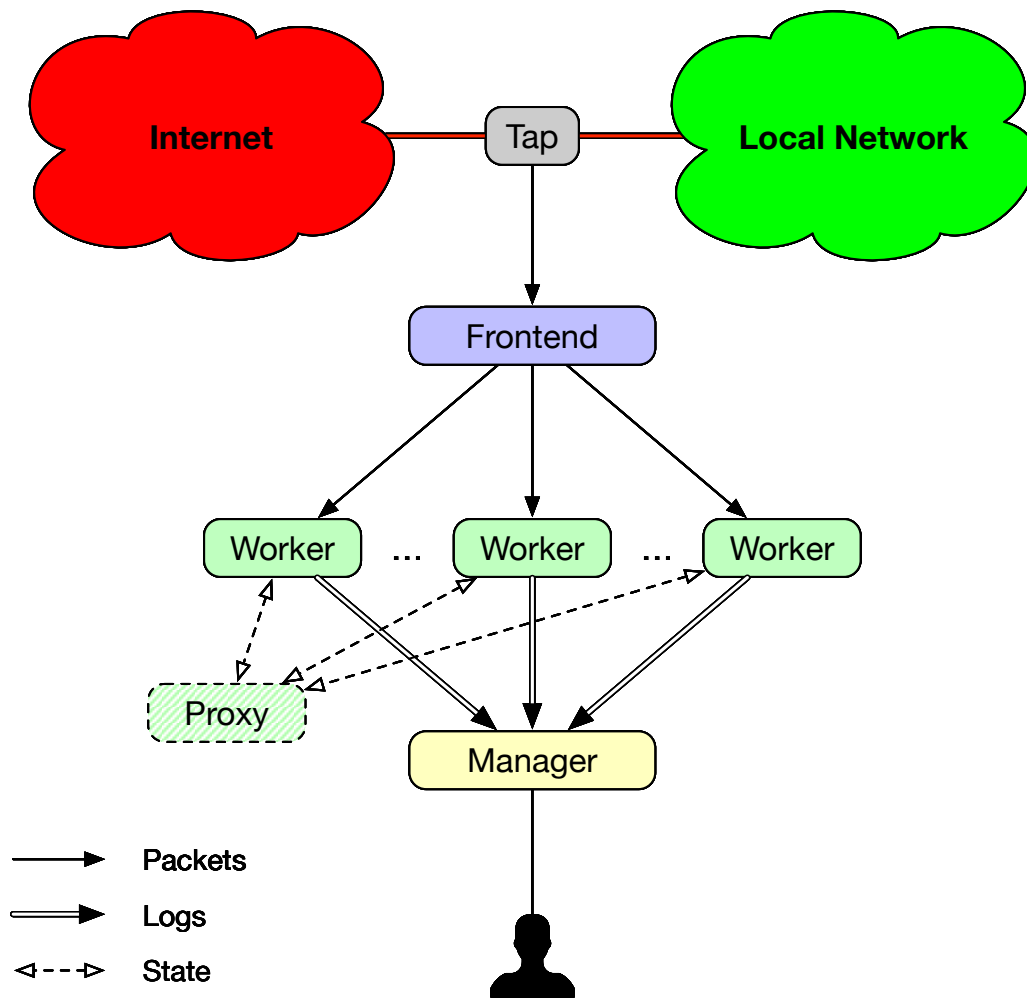
*following def. in: Bjarne Stroustrup, **Software Development for Infrastructure**, *IEE Computer 45*, 2012.

**What is our vision for a
next-gen Bro?**

Deep Detection

- Correlation in multi-hop processing pipelines
- Distribution with pub/sub data access
- Resilience through replicated data stores

Bro Cluster



Vision for a next-gen
Bro with CAF.

#1: **agile** rebalancing via
netcontrol & broker.

#2: **pub/sub & consensus**
instead of shared state.

#3: **fault-tolerance & failover**
through snapshotting.

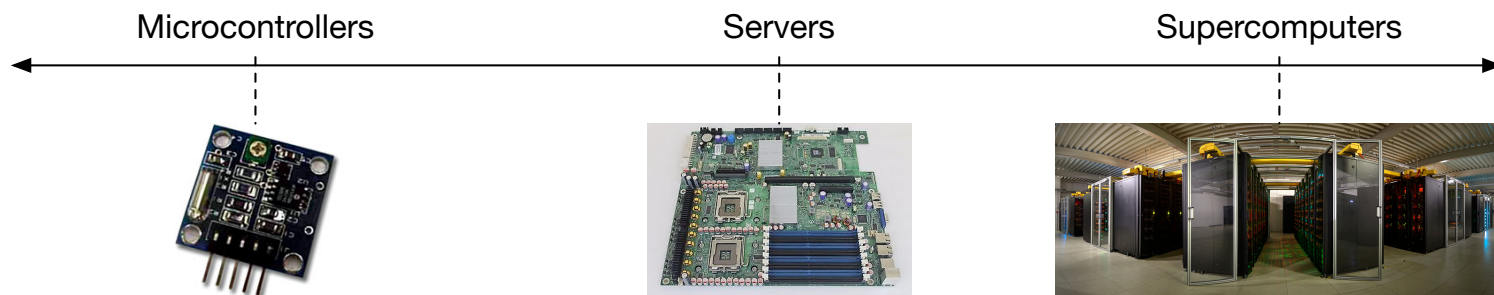
Leveraging CAF

- Bro has to grow with user demands
- Scaling up and out is key to meet future work loads
- CAF provides building blocks for a next-gen Bro

What is CAF, exactly?

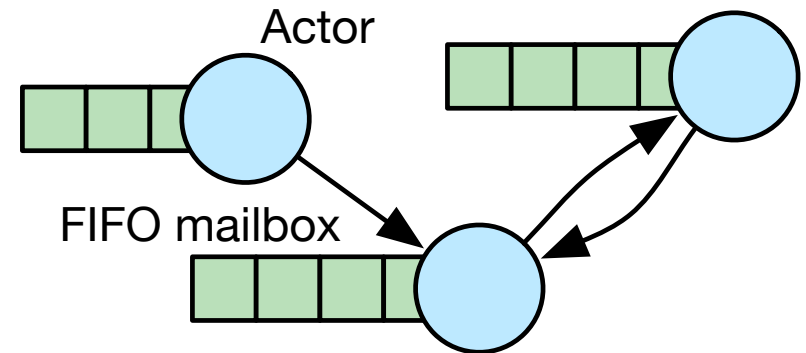
Scalable Abstractions

- **Actors** avoid race conditions by design
- Unified API for **concurrency & distribution**
- **Compose** large systems from small components
- Scale runtime from the **IoT** up to **HPC**

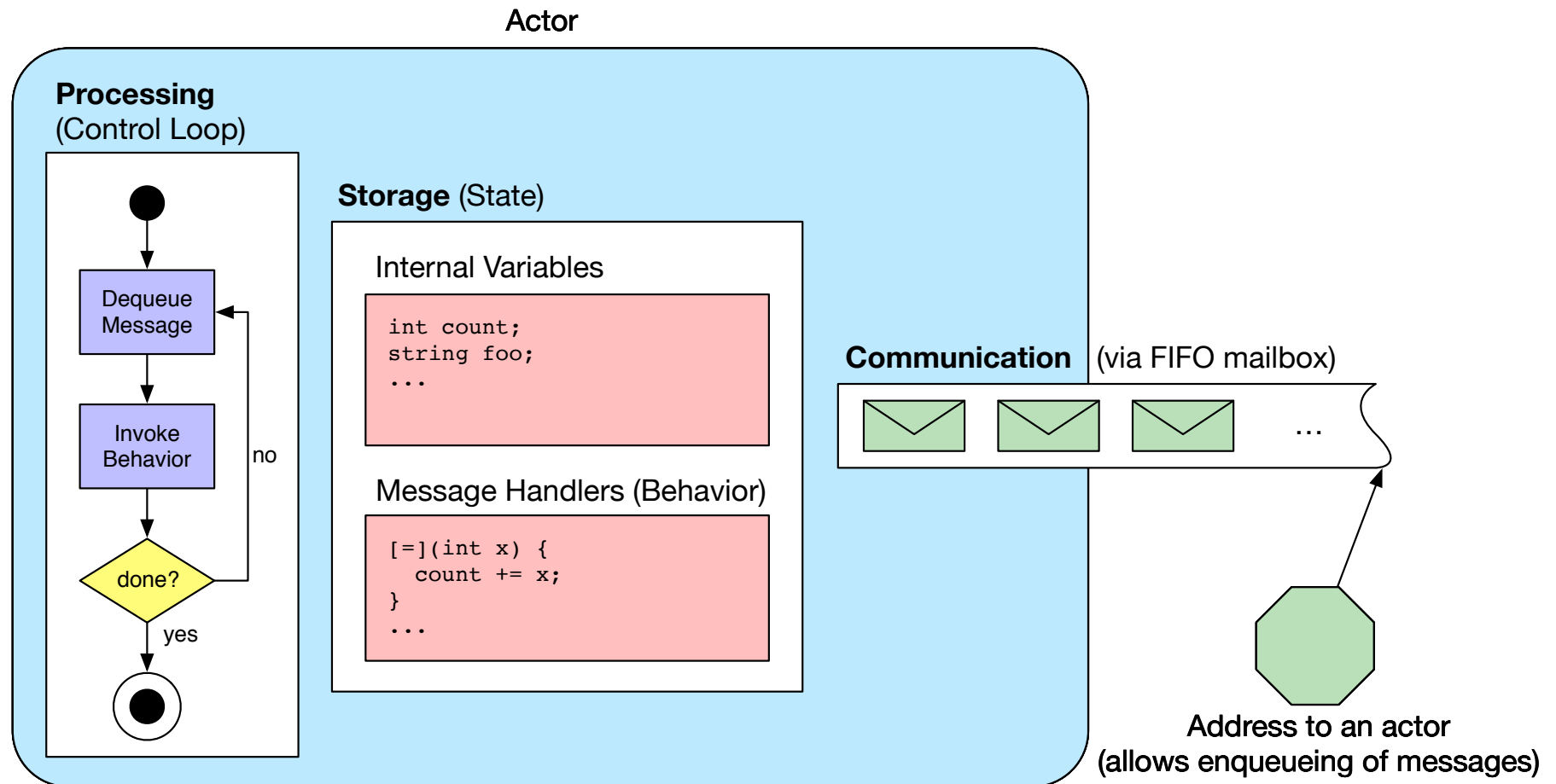


The Actor Model

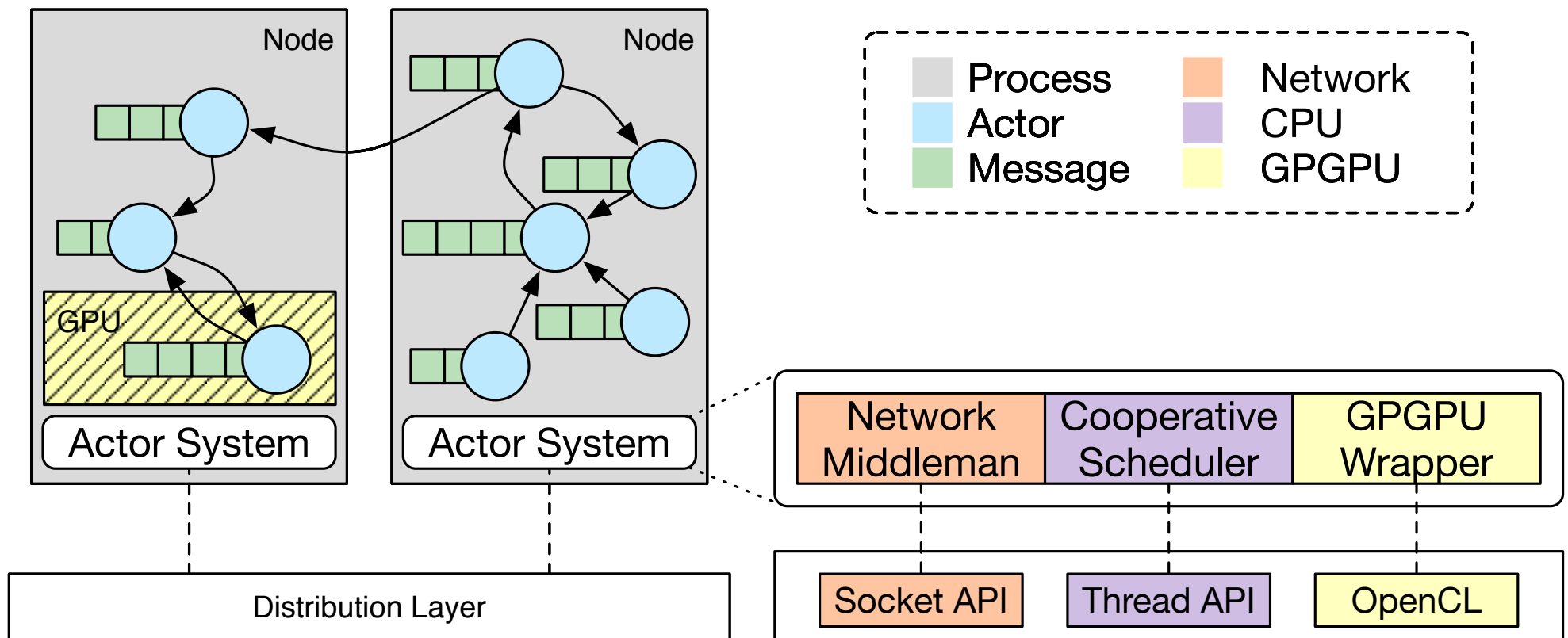
- Asynchronous message passing
- No shared state
- Divide & conquer work flow
- Hierarchical failure handling & propagation



Anatomy of an Actor



CAF's Architecture



Communication Patterns

- CAF offers various messaging primitives:
 - Asynchronous **"fire & forget"** messages
 - **Request/response** messaging (with timeouts)
 - **Pub/sub**-based group communication
 - **Streaming** pipelines (*soon-ish*)

CAF Facts Sheet

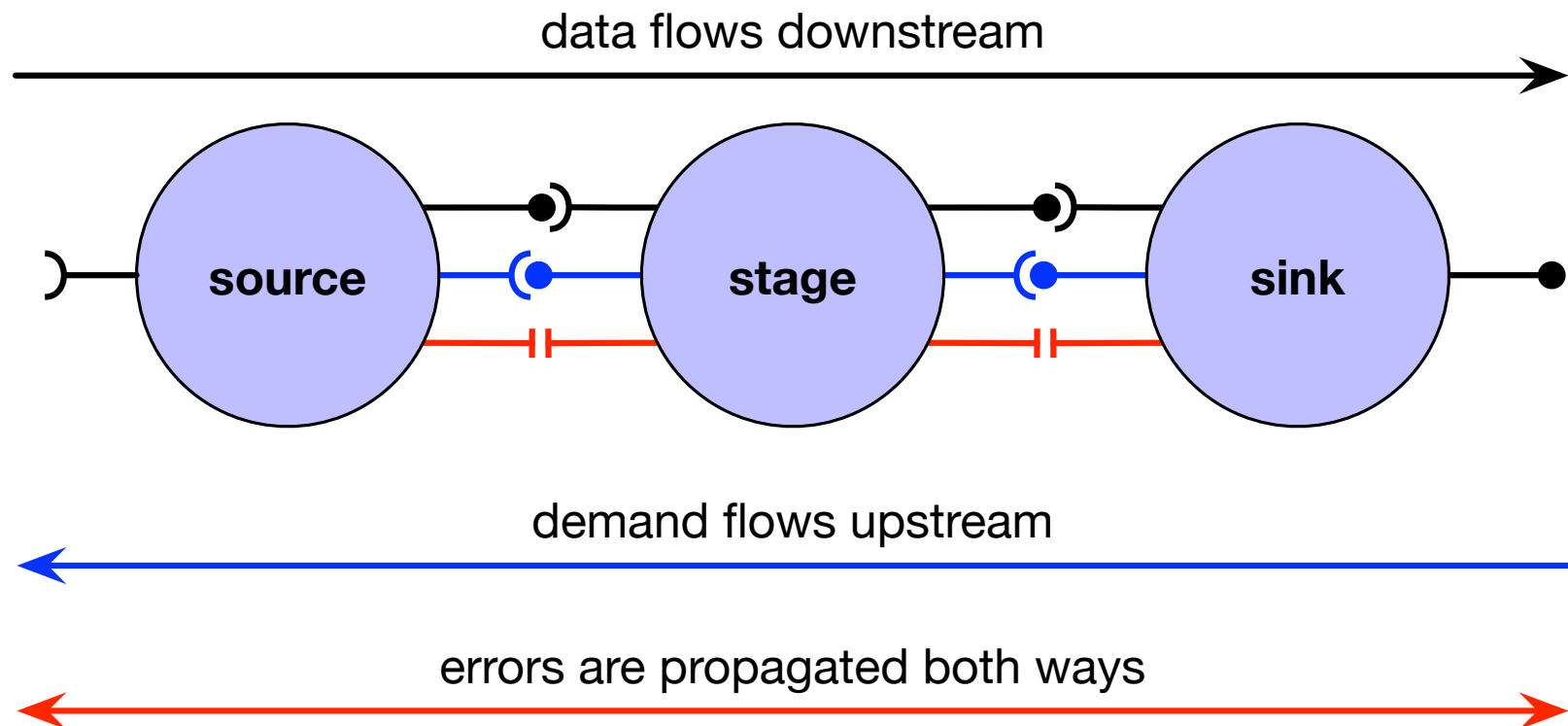
- Developed at iNET research group
- First commit: March 4, 2011
- Active international community
- > 40,000 lines of code (<https://www.openhub.net/p/actor-framework>)

What is next?

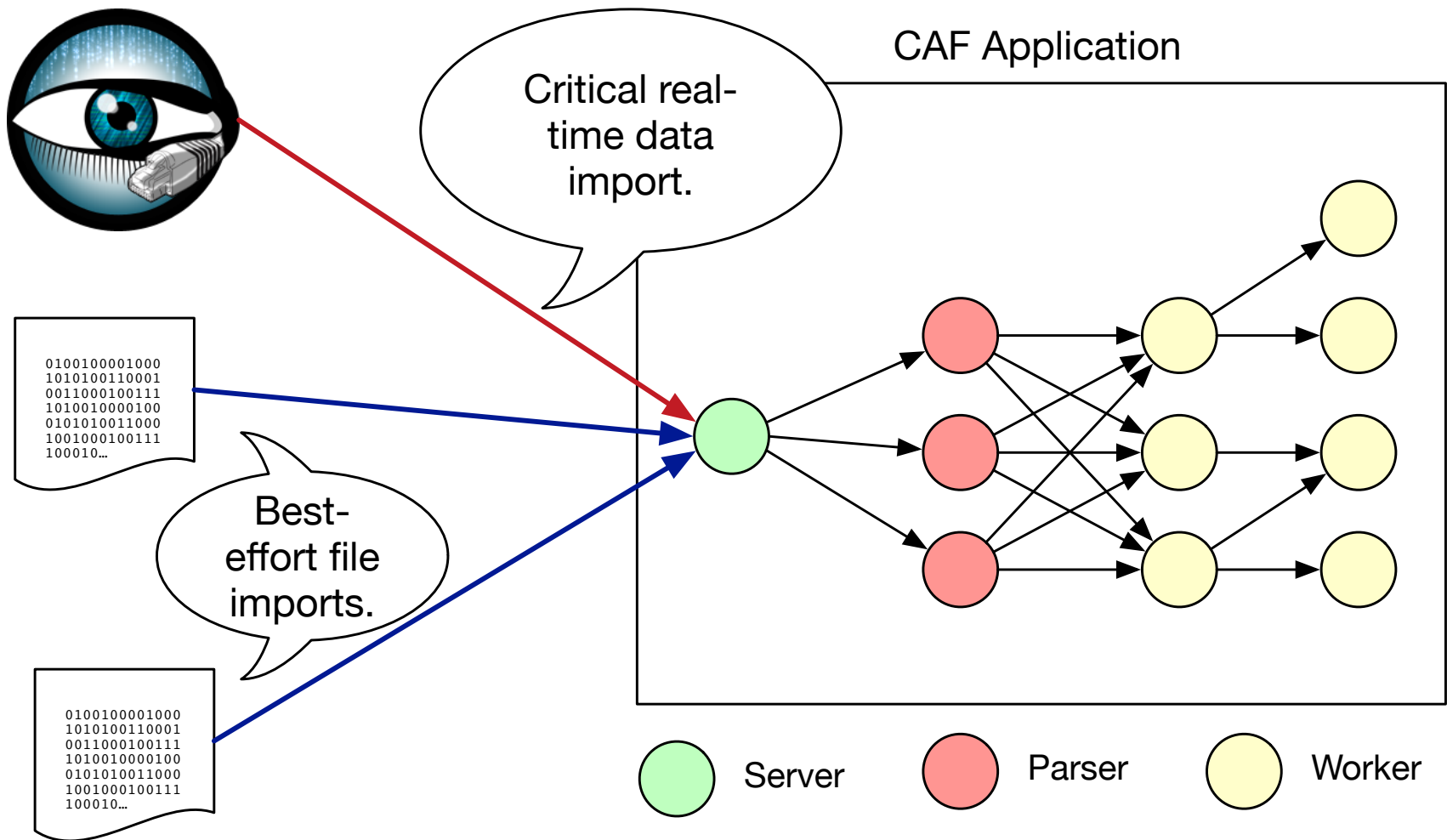
Streaming

- Streams as **first-class citizen** in CAF
- **Priority-aware** message processing
- **Re-deployable** actor pipelines with back pressure

Streaming Concept



Streaming Bro Events



High-level Clustering

- Declarative **API** for deploying actors & pipelines
- Dynamic redeployment & -configuration
- Monitoring of running CAF applications

Debugging Support

- Debugging distributed applications is **challenging**
- CAF's logs can reproduce **causal ordering**
- **Visualization helps** devs understand their system, e.g., with *ShiViz*:

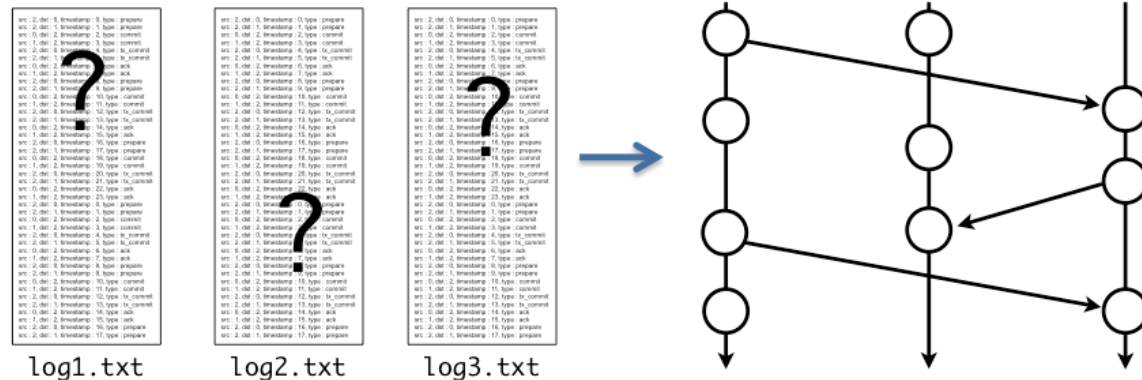
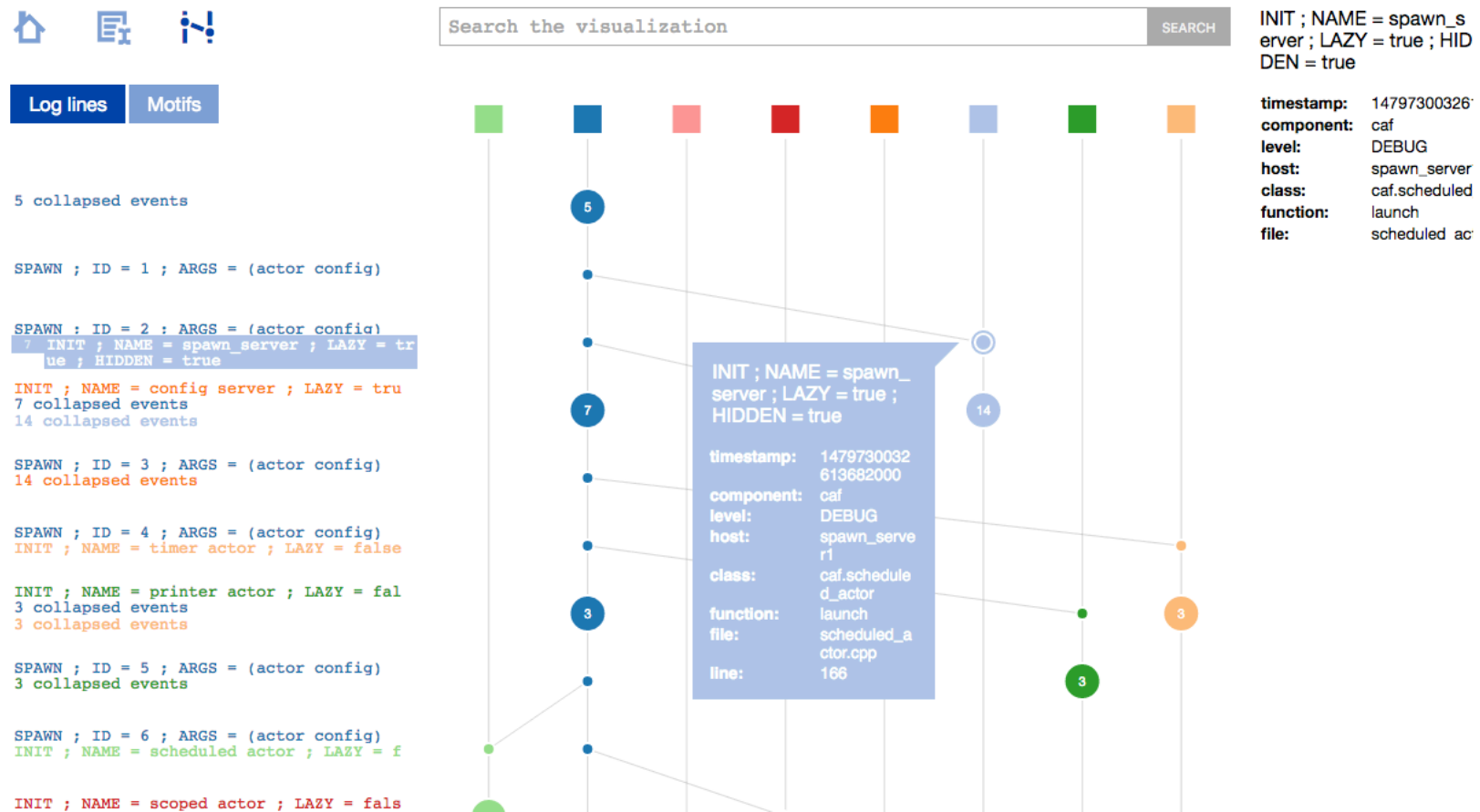


Image source: <https://bitbucket.org/bestchai/shiviz/wiki/Home>

ShiViz* UI with CAF App.

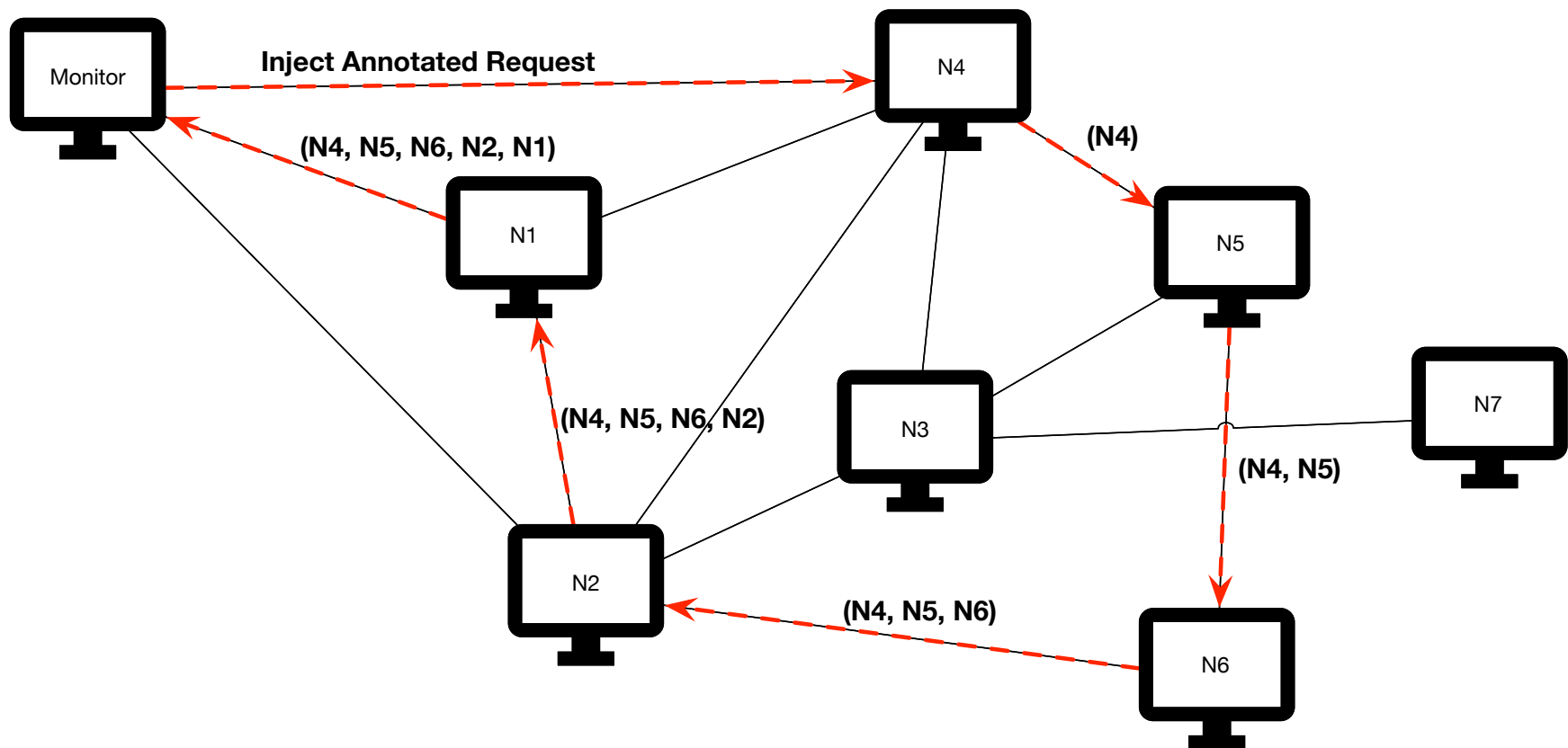


* see: <https://bestchai.bitbucket.io/shiviz/>

Tracing

- **Lightweight** monitoring of data flows
- Captures **causal and temporal** ordering of events
- **Recording** (debugging) or **sampling** (monitoring)

Tracing: Example



Tracing: Visualization

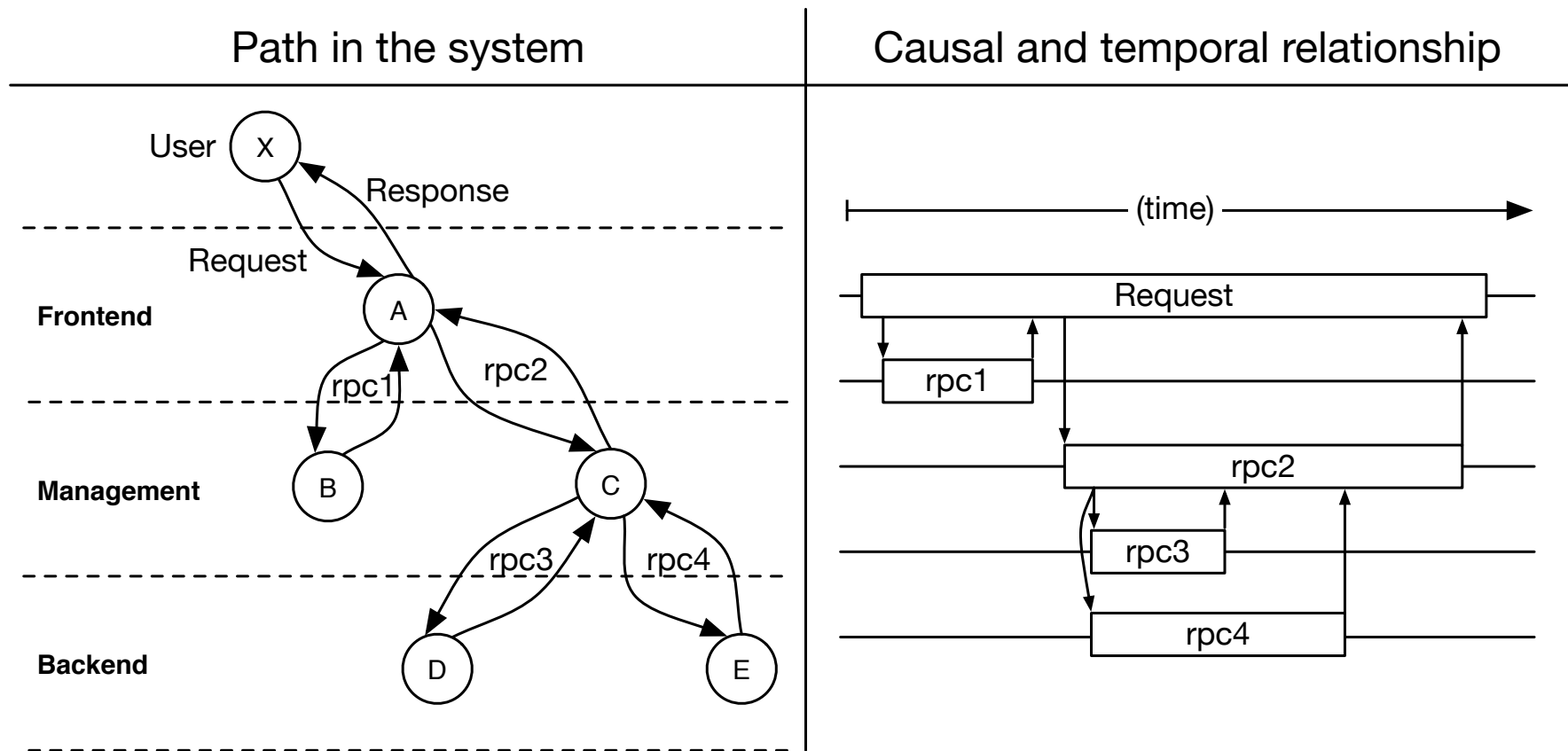




Fig. mod. from: Benjamin Sigelman et al., **Dapper, a Large-Scale Distributed Systems Tracing Infrastructure**, *Google Technical Report*, 2010.

Thanks for Listening

 bro/broker

 actor-framework

 actor_framework