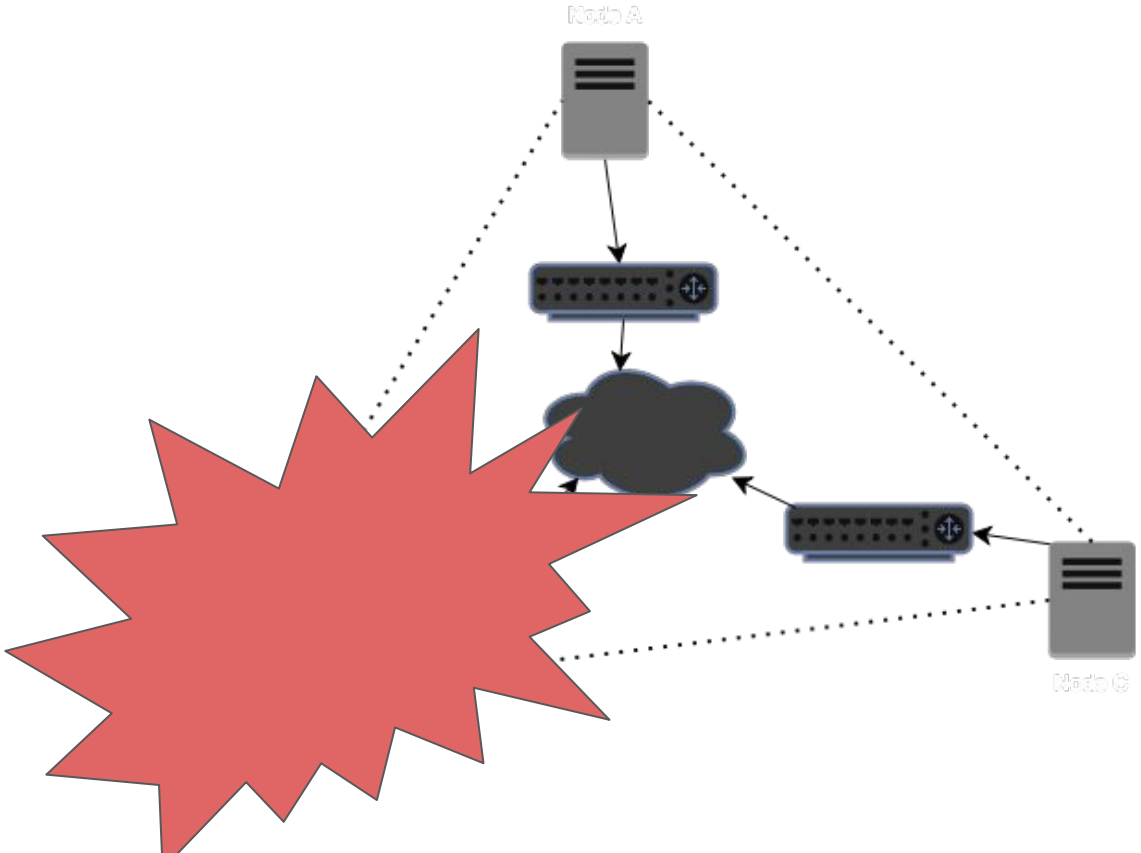
# A distributed file system for OpenBSD

BSDCan 2025

# Agenda

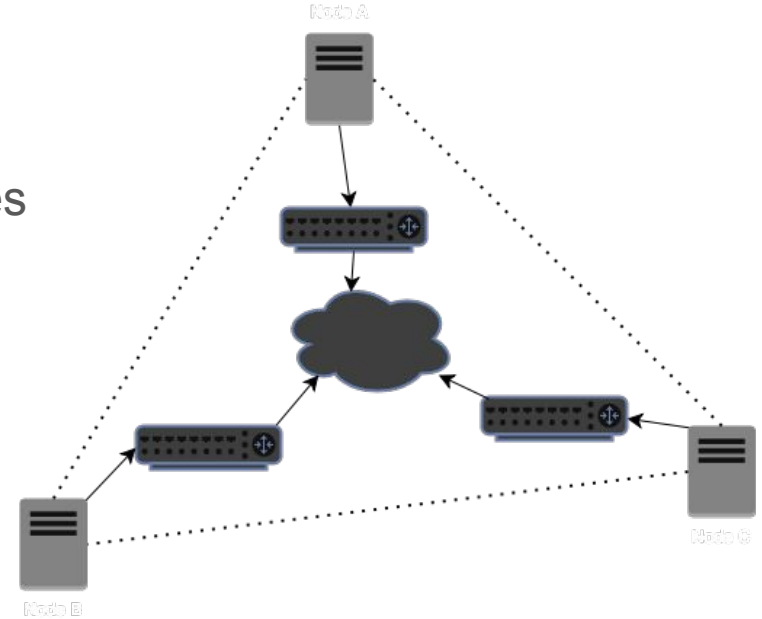
- What’s the point / why does this / what does this do?
- Existing alternatives and why they don’t fit
- FUSE
- Elixir / BEAM
- Consensus algorithms
- Raft
- Demo
- Questions

# What’s the point / what does this do?



# What’s the point / what does this do?

- Ensure files are distributed on all hosts
- No single point of failure
- All nodes have the same software and roles
- Must work on OpenBSD
- No serious performance concerns
    - Single writer is fine



# Alternatives

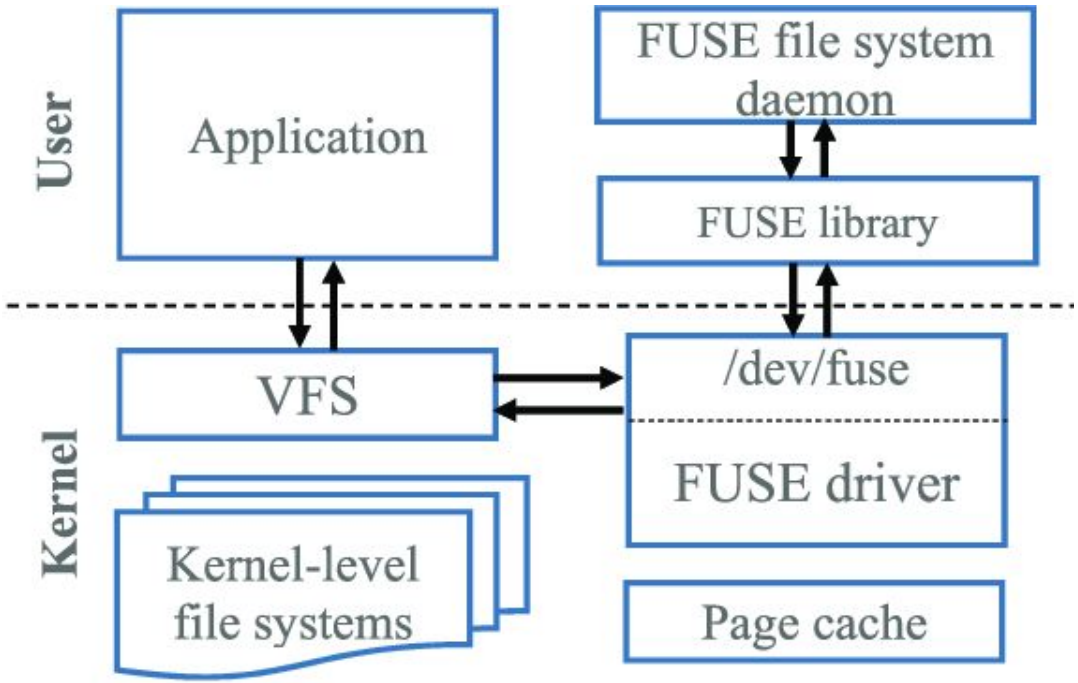
- s3fs+minio / ceph / seaweedfs
  - Don’t work because of FUSE version or OS specifics
- Change of architecture
  - Traditional separate cluster that exposes NFS
- Use a different operating system
  - Run ceph or s3fs on top of vmd
- bioctl + iscsi + carp + nfs
  - Does work; breaks “all nodes are the same”

**Painful; Would not recommend; Works though**

# FUSE - Overview

- Stands for filesystem in user space
- Allows for programs to define filesystems
- Programs define functions for filesystem operations
  - getattr
  - mkdir
  - unlink
  - chmod
  - … etc
- Flexible.
- Not as performant as in-kernel systems

# FUSE - Diagram



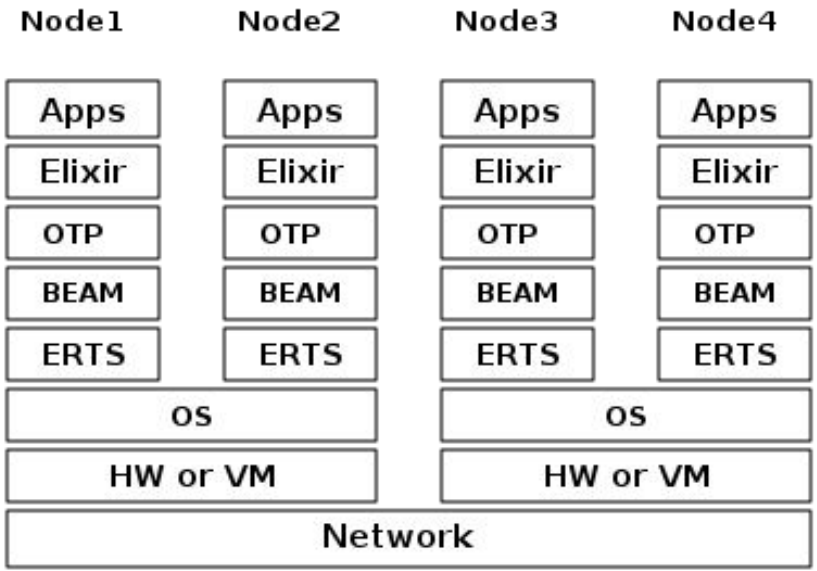
# FUSE - OpenBSD

- Added in 5.4 ( 2013 )
- OpenBSD implementation conforms to FUSE version 2.6
- Some packages exist
  - sshfs
    - Mount a directory from a remote machine using ssh as the transport
  - exfat
    - The exfat filesystem
  - unionfs
    - Overlay directories into a single mount point; Top down check if file exists;
    - Example utilizes unionfs and nfs - reminder that unionfs deals with **directories**

# ERTS / BEAM / OTP / Erlang / Elixir

- ERTS is the Erlang Runtime System
  - I/O, scheduling, memory management
  - Networking, os signaling
  - Aside: RunTime vs Runtime; Ericsson implementation
- BEAM is the virtual machine used with ERTS
  - Register based
  - Analogous to JVM in Java (for our purposes)
- OTP (Open Telecom Platform)
  - Collection of libraries and applications that run on the BEAM VM / ERTS
  - Applications like mnesia (a database), the erlang compiler, networking code, etc…
  - Think JRE without the inclusion of the JVM.
- Erlang and Elixir
  - The language. Others include Gleam, LFE (Lisp Flavoured Erlang), …

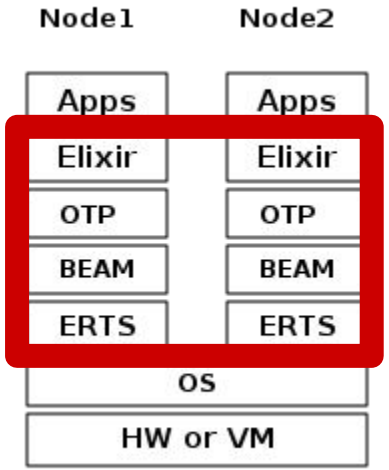
# ERTS / BEAM / OTP / Erlang / Elixir




( Section 1.3.2 from https://blog.stenmans.org/theBeamBook/ )

# For the purposes of this talk

pkg_add elixir-1.18.3




( Section 1.3.2 from https://blog.stenmans.org/theBeamBook/ )

# Ports and NIFs

- Ports
  - Separate programs*
  - Child process from the OS perspective
  - Communicates in binary via async stdin/stdout
  - Cannot crash the BEAM
- NIFs
  - Compiled and linked
  - Same process from the OS perspective
  - Synchronous communication
  - Can crash the BEAM

# Consensus algorithms

Are not discovery mechanisms!

# Consensus algorithms

- PhDs are written about this.
- Really roughly:
  - “The process that is used to ensure that multiple independent systems agree”
- Nuanced and rich in details
  - Need to handle Byzantine fault (unpredictable or actively malicious behaviour)?
  - How quickly (in terms of number of messages) do you need the agreement to happen?
  - Do you need all systems to agree all the time?
  - What is the notion of time?
- There are many ***many*** different algorithms.
  - Paxos
  - Raft
  - Crypto / DHTs / Blockchains - ( Proof of Work, Stake, Burn, Capacity, … )

# Consensus algorithms - Crypto / DHTs / Blockchains

- Same terminology of “consensus algorithm”
- Designed for very large numbers of peers
- Almost always public chains (anyone can join and use the algorithm)
- Responsible for network (in the crypto sense) security
- Mechanism that validates transactions

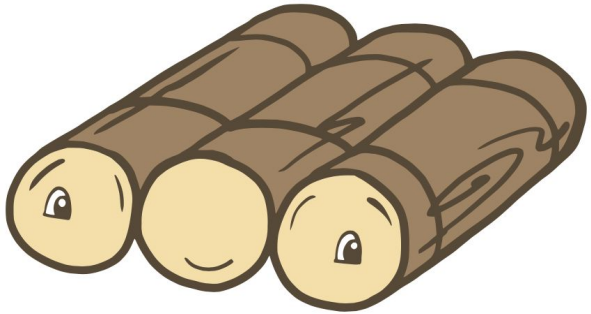
# FLP “impossibility proof”

- Fischer-Lynch-Paterson paper from 1985.
- Relates to distributed consensus.
- A consensus algorithm can only satisfy two of:
  - Safety (agreement/consistency)
  - Liveness (termination)
  - Fault tolerance
- Very roughly effectively states that one of these is required:
  - Failure detection (bounds on time)
  - Eventual consistency (temporary divergence)

# Paxos

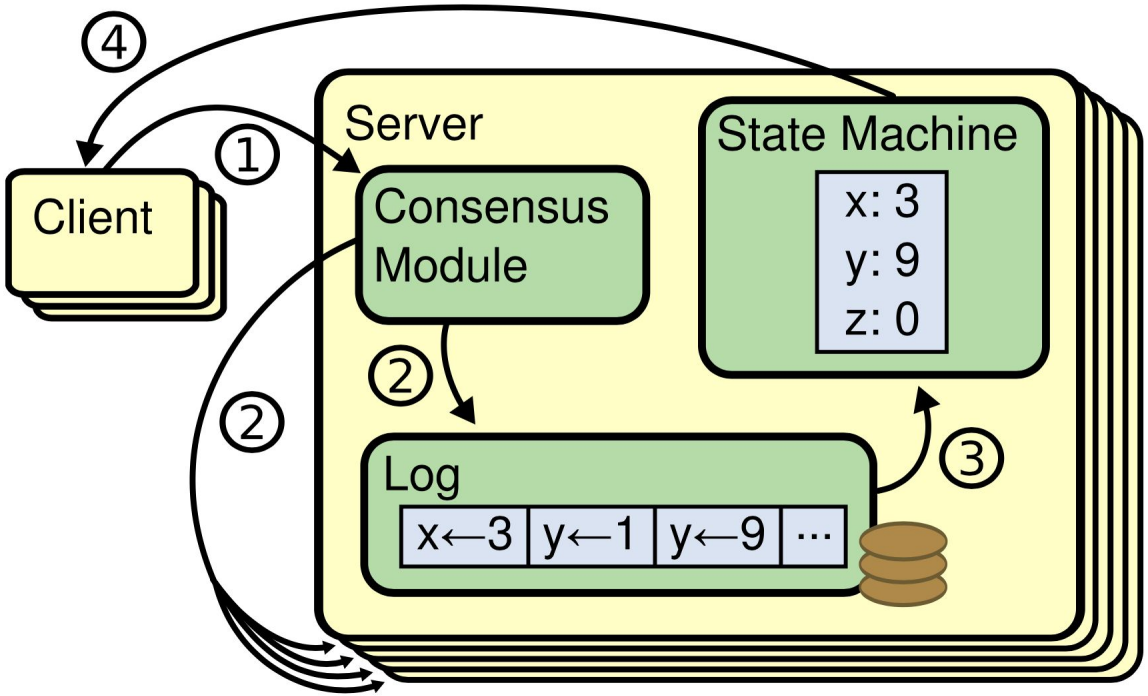
- Tried and true
  - Well studied ( common in computer science curriculums )
  - >25 year history. Conceived in 1989; Published 1998; “Paxos Made Simple” in 2001
- Guaranteed consistency
- High probability of reaching a decision
  - “Provides engineering hints as to what to do to have a good probability of getting lucky”
- Used in many ***many*** systems
  - ceph, google spanner, amazon dynamodb, xtreemfs, etc…
- Original only a single decision/value
- Many different variants
  - multi-paxos, byzantine-paxos, fast-paxos, etc…

# Raft

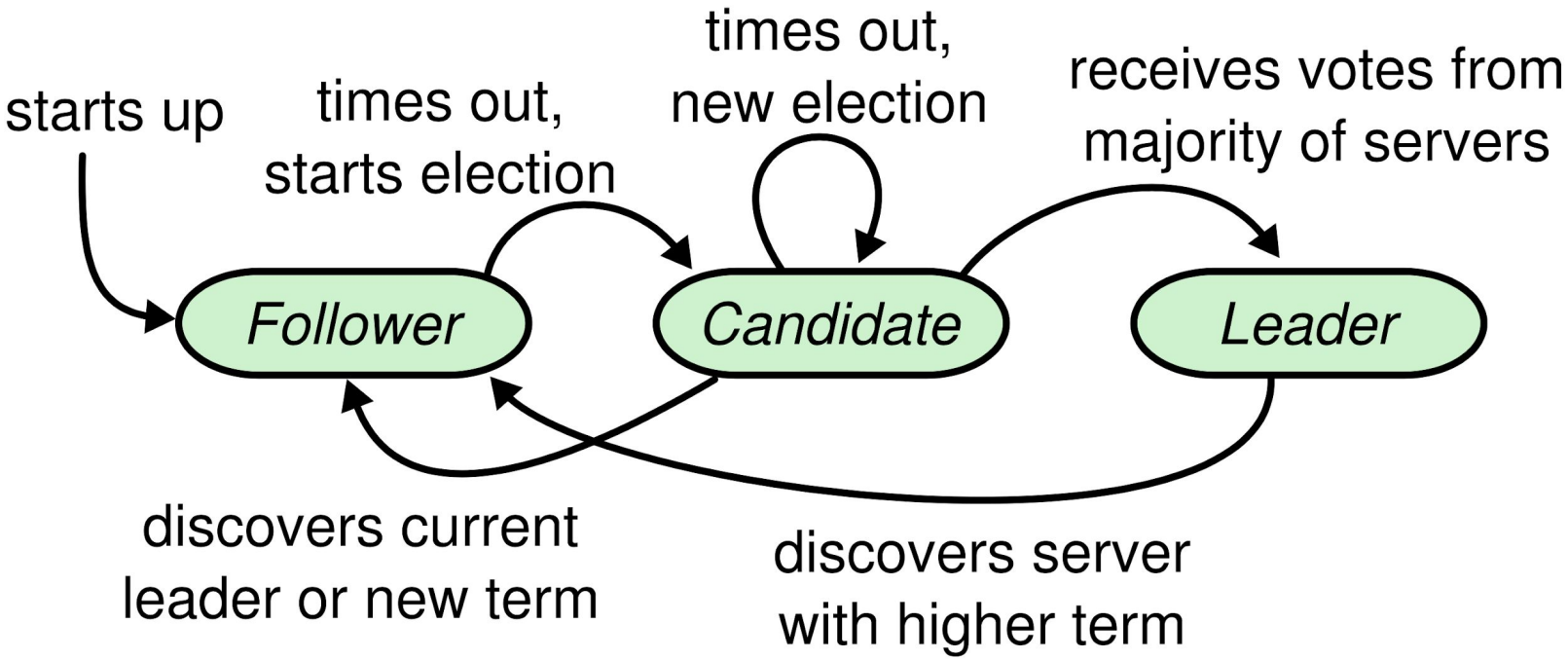


- Designed to be easy to understand
- Has a “strong leader”
- Guaranteed consistency
- Exposes a replicated state machine (and log)
  - Interactions to it appear like a normal state machine
  - Any changes in state (setting a value) have consistency
- Very popular
  - etcd (thus also kubernetes), rabbitmq, kafka, clickhouse, hashicorp stack, etc…

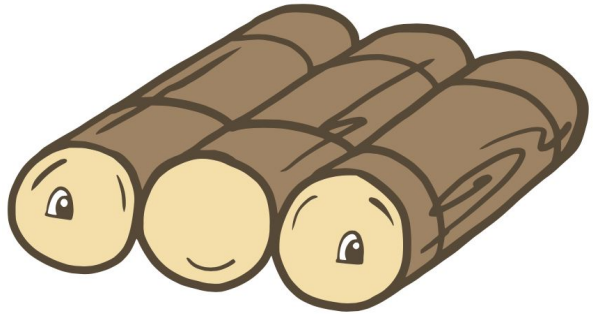
# Raft


④
①
Client
Server
Consensus Module
State Machine
x: 3
y: 9
z: 0
②
②
Log
x←3
y←1
y←9
...
③

# Raft


starts up
times out,
starts election
times out,
new election
receives votes from
majority of servers
Follower
Candidate
Leader
discovers current
leader or new term
discovers server
with higher term

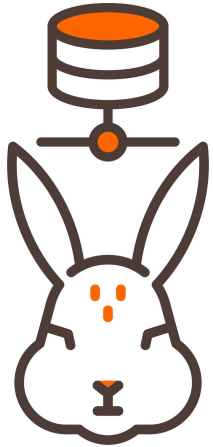
# A few last notes on raft



- Compacting of logs
- Notion of terms
  - One leader per term
- Defines the RPCs
  - AppendEntries (also heartbeat)
  - RequestVote
  - InstallSnapshot (for very behind clients or new members)

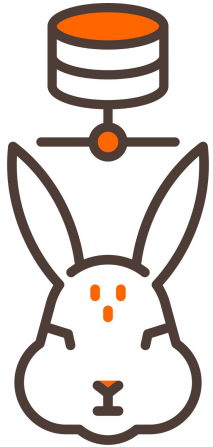
# Khepri

- Replicated on-disk database library for Elixir
- Developed by the RabbitMQ team
- Specifically to transition away from mnesia
- Behind a feature flag in current stable
- Stores the entire contents in memory as well



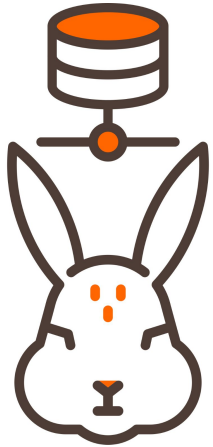
# Khepri

- Storage has bunch of options
  - WAL sizes, entries, etc
  - Write strategies
- Export and import functionality



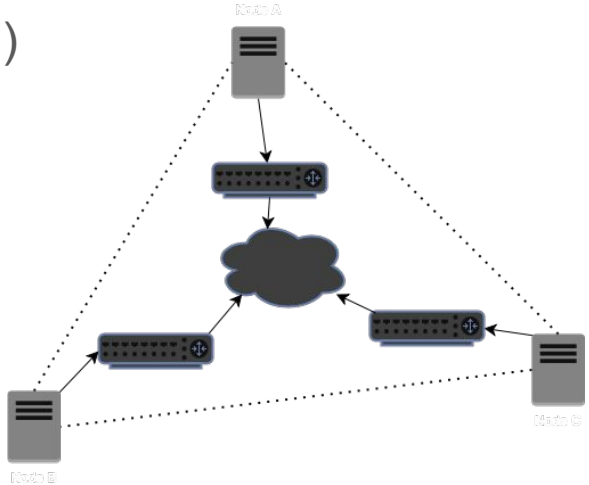
# Khepri

- Tree like structure
- Basic put / get / delete functions
- Transaction support
- Triggers / stored procedures

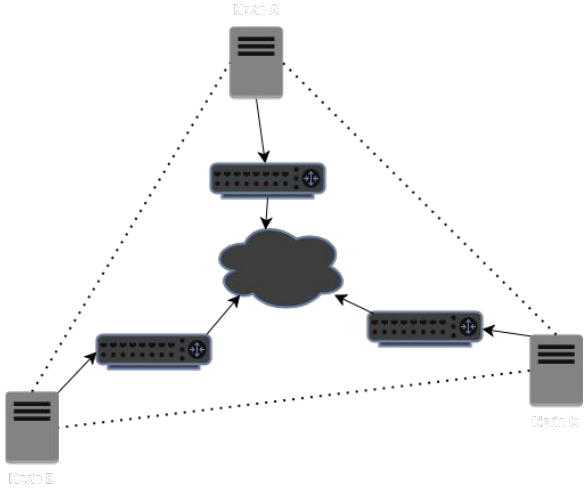


# Pulling it all together

- Three OpenBSD nodes
- Each node runs an Elixir application **and the same code**
- Nodes can find each other (discovery mechanism)
  - static
  - libcluster
- Khepri is started and a cluster is formed
- A C program is started from Elixir
  - fuse_main() is called within the port

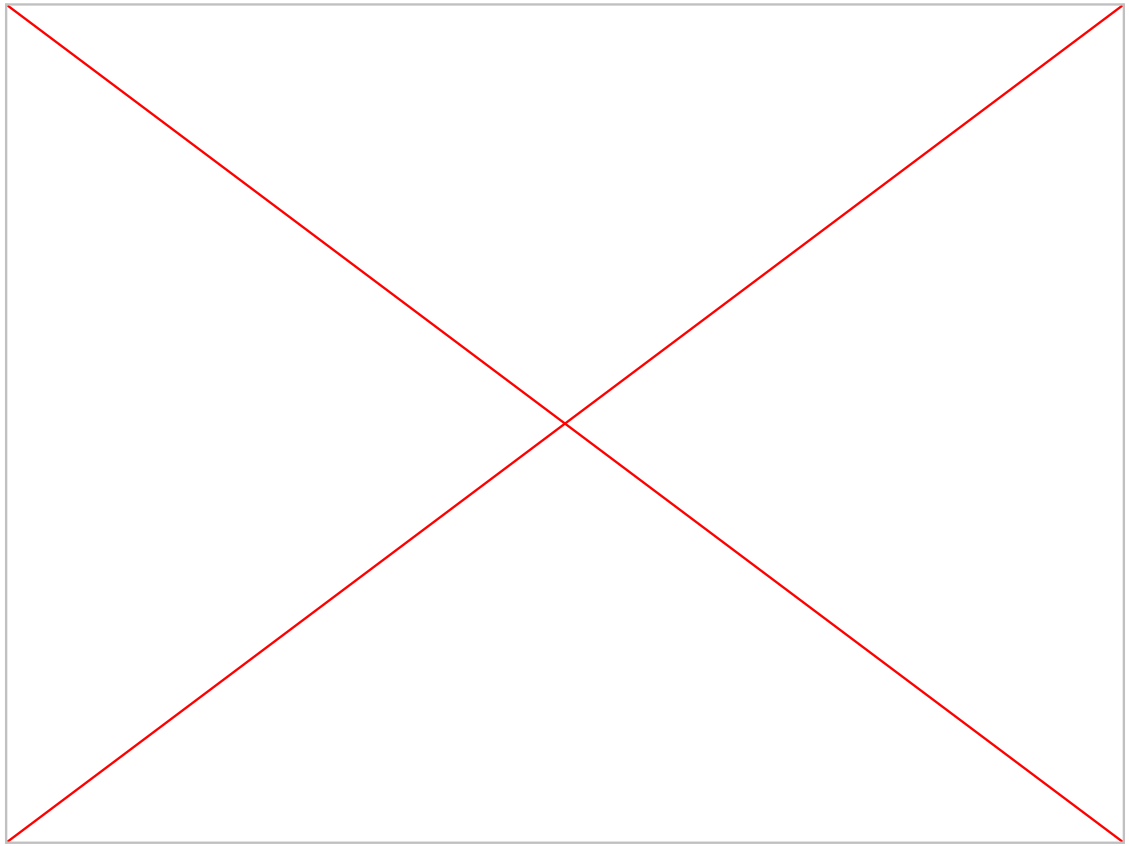


# Pulling it all together



- Single port based package
  - efuse ( efuse and userfs )
  - Modifications to Makefile, include paths, etc.
  - Fix buffer overread and segfaulting for read
  - Implemented the open() call
  - Thanks for the people who maintain ktrace and kdump
- Created fuse_nif
  - Uses rustler
  - Ran out of time
- “Make it work”

# Demo



# Things on the todo list

- Rewrite Elixir port
  - Include write path
  - Add security and drop permissions (ie pledge, separation of BEAM instances)
- Look at using ra directly instead of khepri
  - Duplication of information in memory
- Use the existing filesystem for actual storage
  - Instead of storing in a format that internal to elixir (DETs out of RA)
  - Could provide for dirty reads
  - Easy to integrate existing backup mechanisms
- Bump the fuse version

# That’s it — thanks!

Questions / Comments / Heckles