Starfish OS

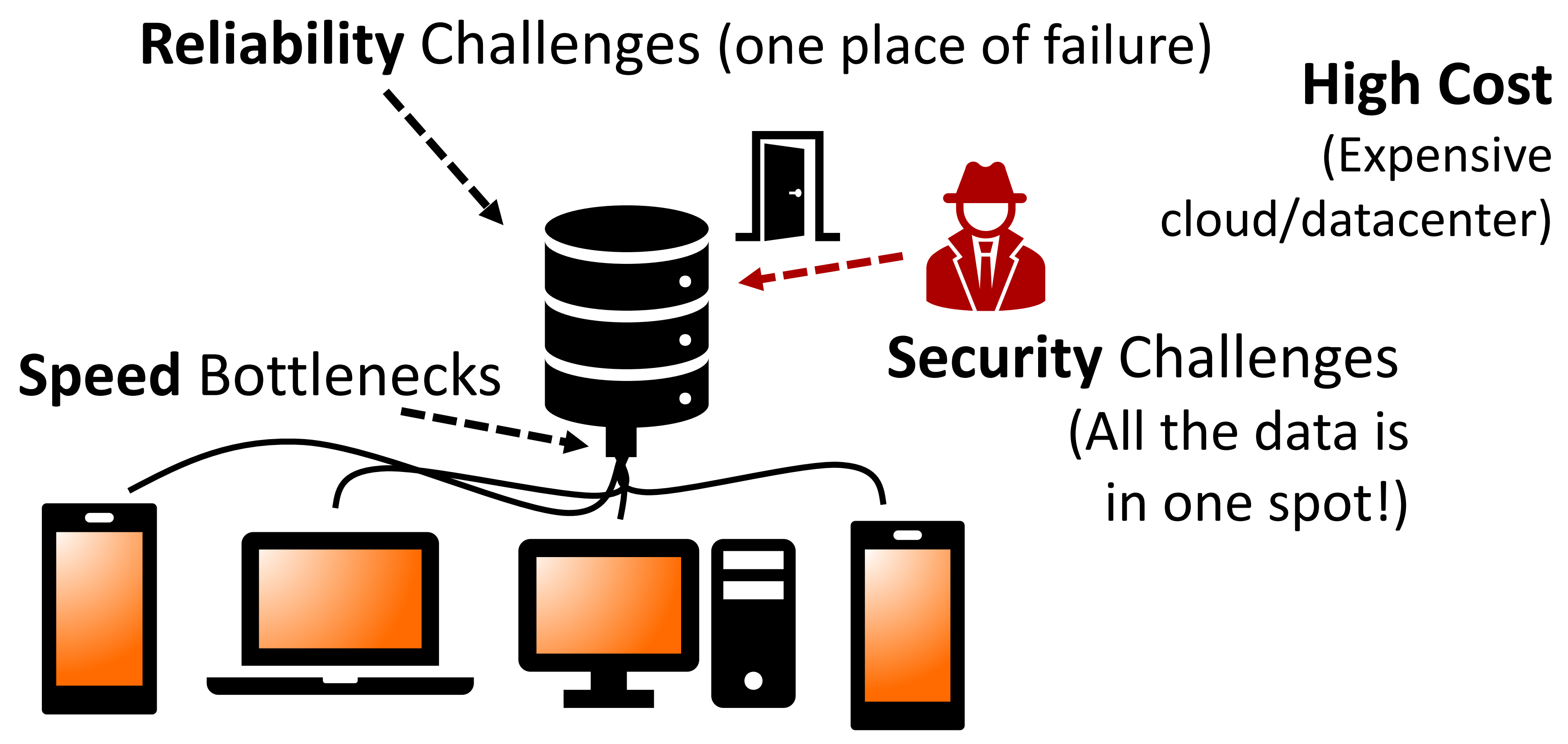
A Decentralized & Distributed OS

Researcher: Benjamin Carter

Advisor: Professor David Demland and Dr. Isac Artzi

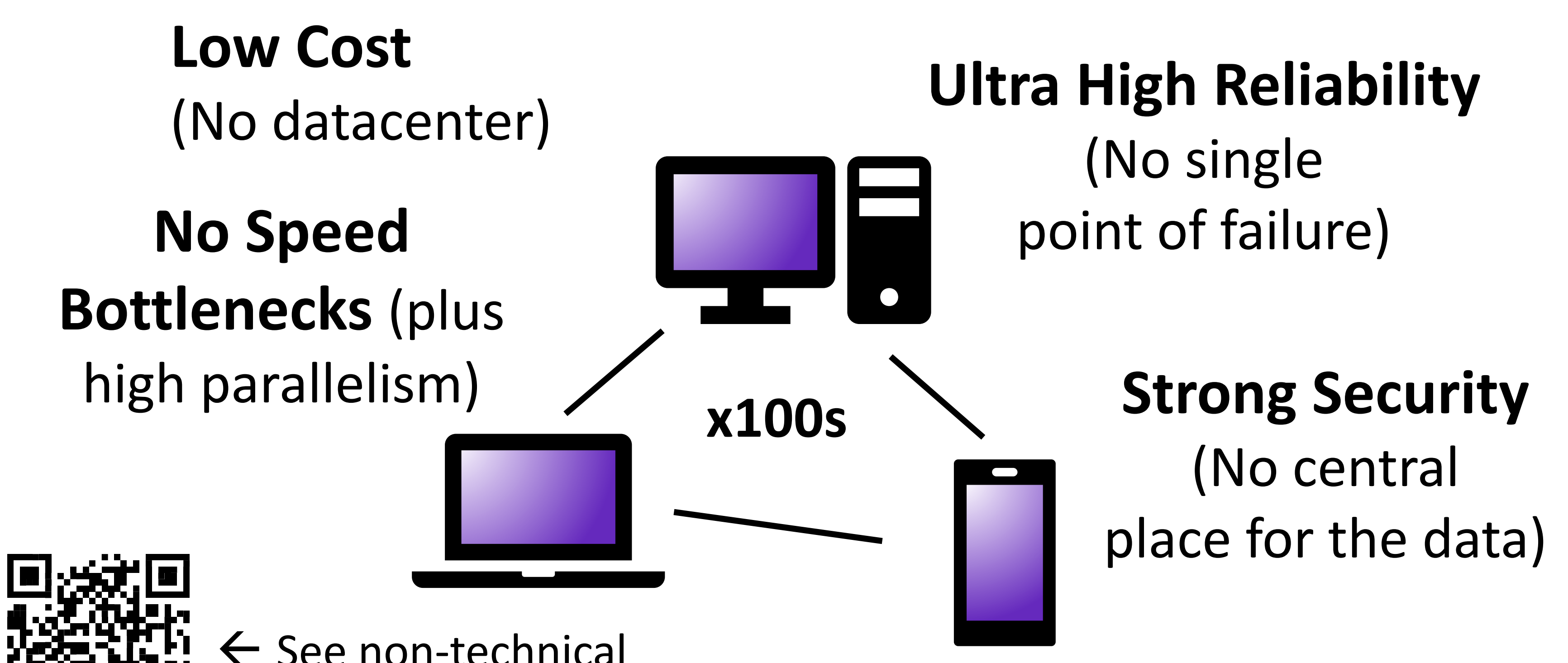
Today...

Technology in our world today is highly centralized.
This poses some problems...
Here below is a typical setup of a website/company

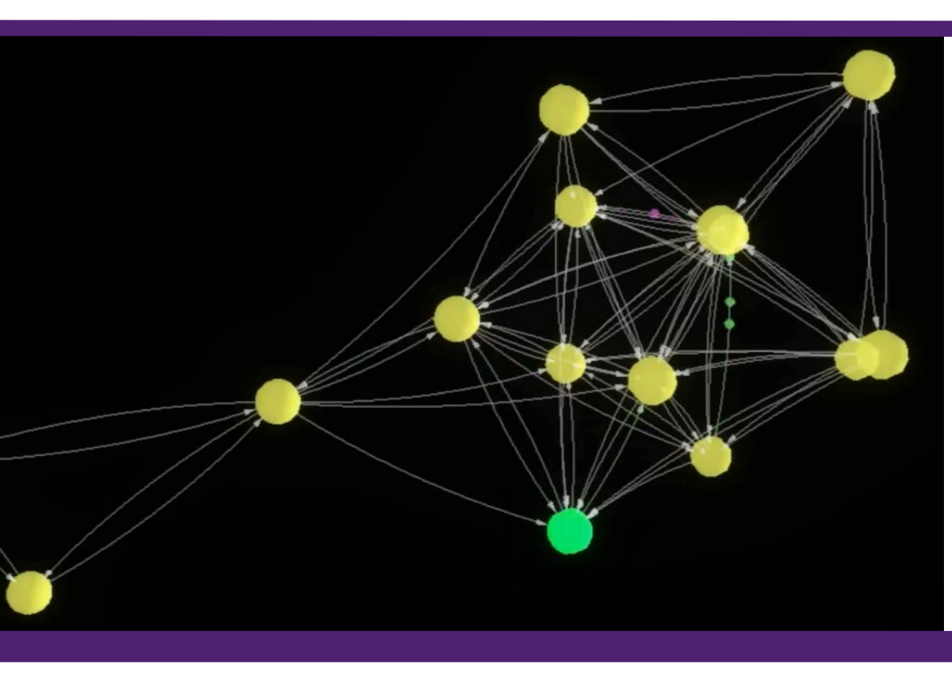

Reliability Challenges (one place of failure)
High Cost (Expensive cloud/datacenter)
Speed Bottlenecks
Security Challenges (All the data is in one spot!)

The world with Starfish

An operating system that is *distributed* and *decentralized* (no central authority) at its core.


Low Cost (No datacenter)
Ultra High Reliability (No single point of failure)
No Speed Bottlenecks (plus high parallelism)
x100s
Strong Security (No central place for the data)

← See non-technical overview presentation here



Simulation All the ideas presented here are represented in a proof-of-concept (PoC) made entirely from scratch.

- Tested on 28 computers. Observed node-dropout.
- Ran several processes including a shell process with Telnet I/O.

See StarfishOS in action!

Displaces need for cloud providers such as AWS. Lowers cost and provides instant deployability

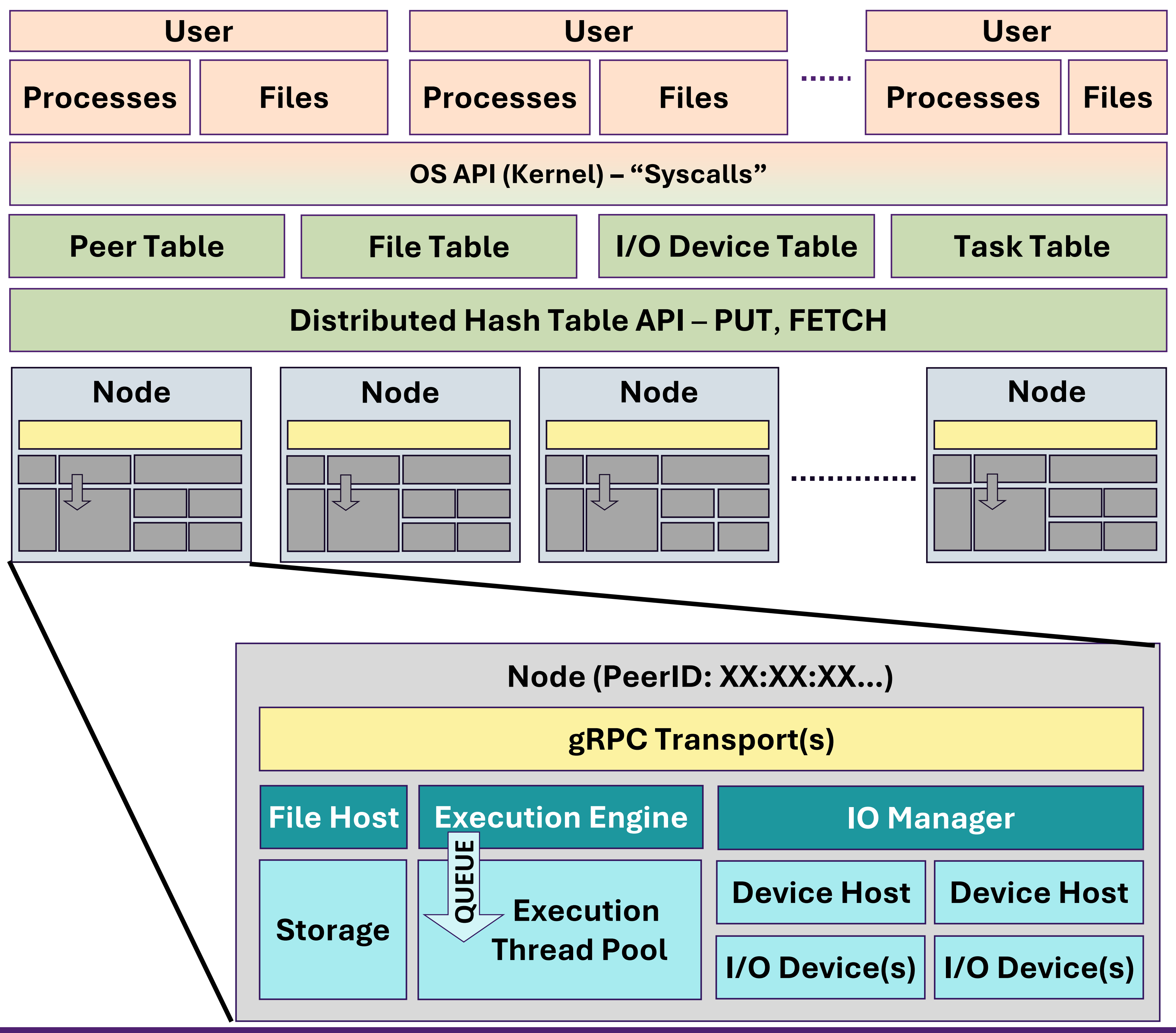
Technical Workings of Starfish OS

OS Architecture

- Distributed Hash Tables (DHTs) are the backbone
- Users disassociated from physical nodes
  - Increases security
    - Anonymous and private computing
  - Increases reliability
    - User not "locked down" to a physical node
    - Renders physical nodes ephemeral
- Processes use syscalls to access kernel
- Nodes communicate using RPC messages
  - RPC DHT Update/Fetch
  - RPC Direct communication
- Node Sandbox (zero-trust node design)
  - Execution Engine
  - Raw file block storage
  - I/O Host

Peer Management

- Nodes can directly connect to one another
- Nodes discover one another through the Kademlia algorithm (used by libp2p)
- Nodes register keep-alive callbacks that trigger kernel events when a node leaves (sending updates to the DHTs)
- Nodes publish their "transport address" on the Peer DHT



Event-Based Processing

I redefined what a process and task is:

A process is a collection of tasks that can pass events to one another, akin to a "neural network" but with processing. It has an initial "start" event.

A task is a stateless set of instructions that receive events and send events. State is carried in the event data. These tasks are stored across nodes in the OS.

An event is a message that carries the state between tasks and contains routing information to find the next task

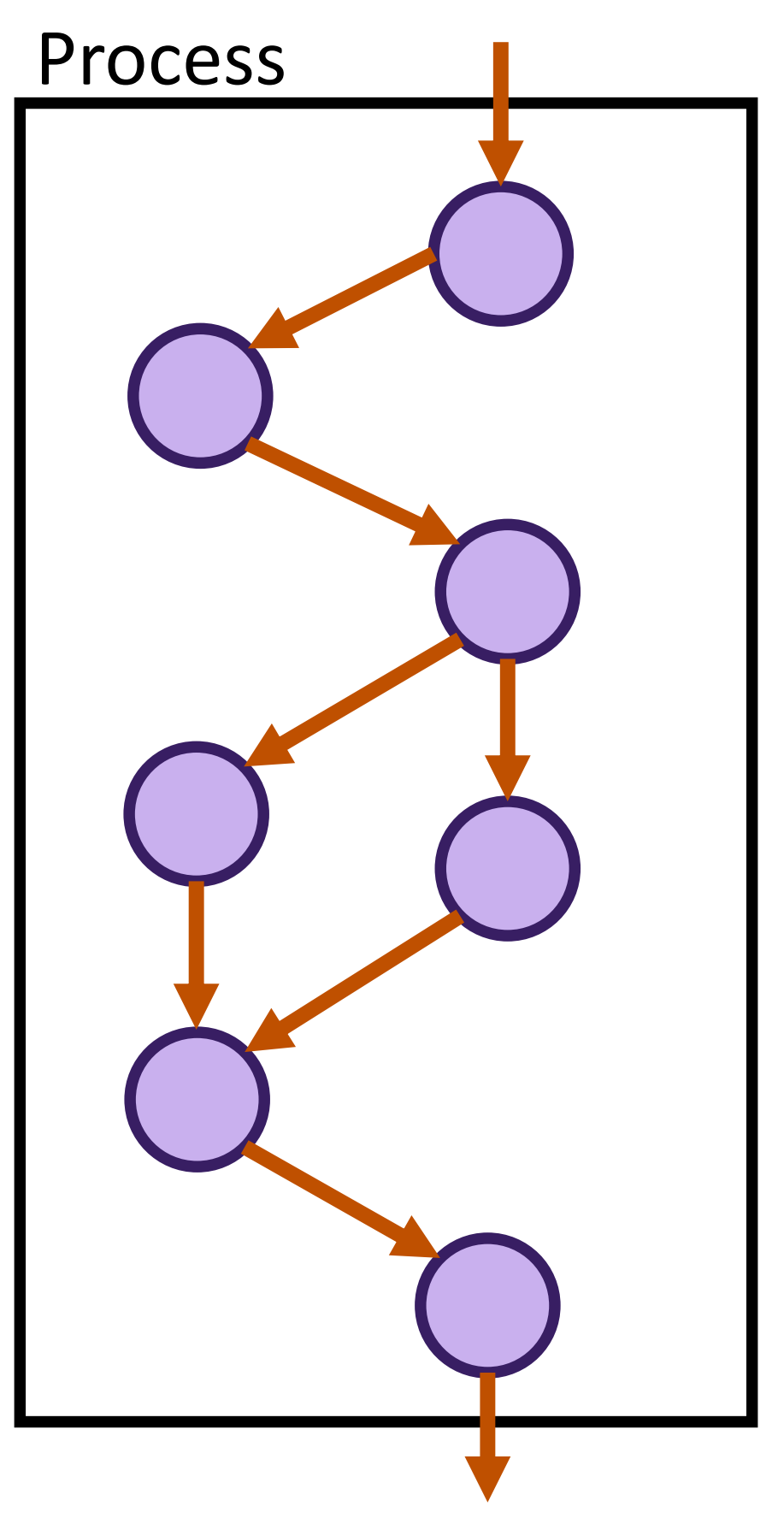
Dropout. Events are journaled and replayed if node holding a task drops. Future will have auto-scaling.

Implications:

This OS is ***memoryless!***

A large shared memory does not scale to 1000s of devices. Large states can be stored in the filesystem.

This OS ***natively supports parallelism!***



Task = Circles
Event = Arrows

Native Distributed Filesystem

Native: to the user, it looks like any other OS, the file system is distributed transparently to the user.

Storage:
Files are broken into blocks and stored across nodes with block addresses in the DHT.

Dropout:
Each file block is stored on a main node and a "monitor node". If either go down, the other respawns the file. The monitor node mirrors all changes from the main node. In the future, this will be made to be auto-scaling on demand.

Native Distributed I/O

Native: to the user, I/O devices are distributed transparently.

I/O: A node receives a device connection, alerting the OS. The I/O device is addressed by the host node, so no dropout support is required.

I/O manager: Sends/Receives OS RPC calls

Device Host: Acts as the server for outside I/O connections

I/O device: The external device connected.

More info about Starfish OS

My Website & Contact Info

Benjamin Carter

BS in Computer Science with Emphasis in Big Data Analytics

Completed Spring 2025

OS Tech: Python Docker gRPC

Sim Tech: three.js *3d-force-graph by Vasturiano* MQTT