Node.js Diagnostics Summit

Ottawa Feb 12th-13th 2018

Schedule

Day 1 (12th)	
Key Themes Part 1	
8:30 - 9:00	Arrive and settle in
9:00 - 9:10	Introductions
9:10 - 9:20	Agenda review
9:20 - 9:30	End user perspectives (Netflix)
9:30 - 10:00	Demo from James/Matteo/Andreas
10:00 - 10:30	Break
Key Themes Part 2	
10:30 - 11:00	Post Mortem - priorities/issues
11:00 - 11:15	Debugging - priorities/issues
11:15 - 11:30	Platform neutrality/APIs
11:30 - 12:00	Testing and Documentation
12:00 - 1:00	Lunch
1:00 - 1:30	Monitoring - priorities/issues
1:30 - 3:00	Breakout 1
3:00 - 3:30	Break
3:30 - 6:00	Breakout 2
6:30	Dinner

Day 2 (Feb 13th)	
9:00 - 11:00	Breakout 3
11:00 - 1:00	Breakout 4
1:00 - 2:00	Lunch
2:00 - 3:15	Breakout review
3:15 - 3:30	Break
3:30 - 5:00	Breakout review continued. Recap & actions.

Agenda

Post mortem diagnostics

- Post Mortem Tooling
    - Themes
        - Current tools (mdb_v8, llnode) get broken when v8 changes
        - Are current tools at correct “level of abstraction”?
        - llnode update: https://docs.google.com/presentation/d/1O6C7a-badKcl5mKmaD0cLUOWDhBUbWXUhV76G4_zEQ8/edit?usp=sharing
    - Post-mortem debugging scenarios
        - what problems do people need to solve?
        - High-level vs low-level
            - is a JS view sufficient (JS heap, JS stack)
            - or do we need node stuff (native buffers, work queues)
            - or even lower? (libuv, threads, raw memory?)
            - are core dumps sufficient? Or are detailed execution traces needed?
    - Desired Outcomes
        - Understanding of scenarios and priorities for post-mortem debug scenarios
        - What are key tools
            - Integration
            - Testing
            - Docs

            - usability
        - Identified plan/priorities on where are the seams between node, JS engines, and tools
        - How do we ensure tools stay up-to-date?
            - Is there a distinction between "user-mode tools" & "system tools"?
                - Are llnode & mdb usable by your average Node.js developer?
                - Are there things that can be done to simplify them?
            - Do some tools need to be part of “node core”
                - E.g., “node-report” - https://www.npmjs.com/package/node-report
        - Promises
            - What do we do about unhandled rejections?
            - Some data on whether the catch prediction in V8 is useful for this purpose.
    - Discussion
        - scenario review
        - short term plan
        - current status of tools?
            - Review existing tools
                - LInode, heap dumps, node report,
            - how do we keep them from being broken?
            - Promote some to “official tools”?
        - long term plan
            - protocol/JS API to support JS-level post-mortem analysis
                - e.g.
                    - CRDP extension?
                    - https://github.com/nodejs/post-mortem/issues/33
                - supports variety of tools written to protocol, adapter to translate between VM's core dump & protocol messages
                - provides hook for multiple VM support (different V8 versions or different engines)
        - Survey to users?
        - Can we use existing tools/debug protocol over core dumps
        - Others?

- Supporting Promises/async functions
    - (some of this is captured above as well)
    - https://github.com/nodejs/post-mortem/issues/16
    - Themes
        - supporting unhandled rejections in core dumps

- Desired Outcomes

Monitoring

From intro, how to feed data from both native and javascript efficiently to allow collection by tools.

1. Asynchronous context tracking
    a. A Model for async context

    b. Async Hooks
    How can we reduce overhead?
    What else is there to do, to leave experimental state?
2. Ability to add before/after hooks instead of wrapping
3. Proposal: Ability to patch module local functions
4. Adding Node.js trace points (trace events)
    a. Faster trace_events logging, especially from JavaScript
    b. Trace_events becoming signal-safe (or even better, hard-kill safe)
    c. Flight recorder
    d. Define criteria for trace-events exiting experimental
5. Faster stack traces
    a. New Error().stack becoming faster or something equivalent
6. Loader Hooks?
    a. Access to module metadata
        i. Associated package.json
        ii. Access to module cache
    b. What kind of loader hooks do we need?
    c. What are the use-cases? Why is the hook needed.
7. Metrics
    In general, right now each vendor has their own ways to collect metrics which makes them hard to compare or quantify. How can we create a vendor-neutral collection of standard metrics?

    a. EventLoop Metrics
        i. API for access to metrics
        ii. Which ones are relevant?
    b. Outcomes:
        i. What metrics do we want exposed from node core?
        ii. What metrics we need from libuv?

iii. What API changes are needed for better supporting metric collecting tools?

8. Profiling
    a. CPU
    - “Full observability”
    - Granularity of Native frames? (brought up by netflix / google)
    b. Heap
    c. V8’s sampling heap profiler
    d. Other ?
    e. Outcomes:
        i. What is the plan to make these tools better?
        ii. What is needed to make sure this is tested / documented?
        iii. What are the use-cases?

9. What else should Node.js have in order to support great monitoring
    a. Limit undesired side-effects
        i. Unhandled Exceptions, Signal Handlers, "missing" Promises…
            1. Installed listeners should not change application behavior
        ii. Reading stack from Error object modifies object, causing call-sites only available for 1st caller
            1. Strict mode limitations to get references to function objects from call stack

    b. OOM Handling: Add a way to catch or report OOM
    c. Always support multiple consumers on diagnostics APIs like GC
    d. Reduce mem consumption of mem dumps
    e. Reverse lookup from file-line inside function to file-line of function beginning
    f. "Agent mode": access to node internals for "privileged" (diagnostic) modules
    g. How can we improve the production readiness of V8 APIs in Node (e.g. CPU sampling)
    h. Outcomes:
        i.

Outcomes

-
- API changes for VMs to make async hooks faster
- Use cases for async hooks
- Model for async execution/context
    - Gaps between model & async hooks
- Abstraction over Async Hooks to solve async context?
- Plan for Monkey Patching in general
- Plan for trace events

- Plan for faster stack traces
- Ideal API for error object
    - Requesting Stack has side effects on ErrorObject
    - Event when error is used
- Plan to get trace events out of experimental
- Loader Hooks
    - Use Cases
    - What do we need
- Metrics
    - List of metrics
    - APIs (at node and libuv)
        - Metrics at libuv requires abi change
- Profiling
    - Tools, testing, use cases
- What else?
    - List of what node needs to make monitoring great

Notes:

- Async Model
    - Async Hooks should consider model before going out of experimental
- Getting async hooks out of experimental
    -

Debugging

- Async Program understandability
    - Sync stack traces don't provide full understandability
- Filtering?
- Same tools that work for live process as post-mortem
    - LInode adapter for debug protocol
    - Or devtools understanding core-dumps?
    - Time-travel debug may be very relevant to this
- Time travel debug more generally available?
-

Documentation/Guidance

1. Module developers guide (as discussed in VM summit in Germany)

2. How to investigate common issues

Outcomes:

- Issues in our doc approach - alternate approaches (php docs good example)
- Doc matrix
- Exiting doc gaps
- Production v/s Dev tagging

Testing

1. We don't necessarily have good testing or visibility into the testing we do have

2. Diagnostics test pass/fail should be a release criteria ?

Outcomes:

- Validation, use cases
- Validation matrix
- Tools
- Key diagnostic tooling - Release blocking criteria
- Performance

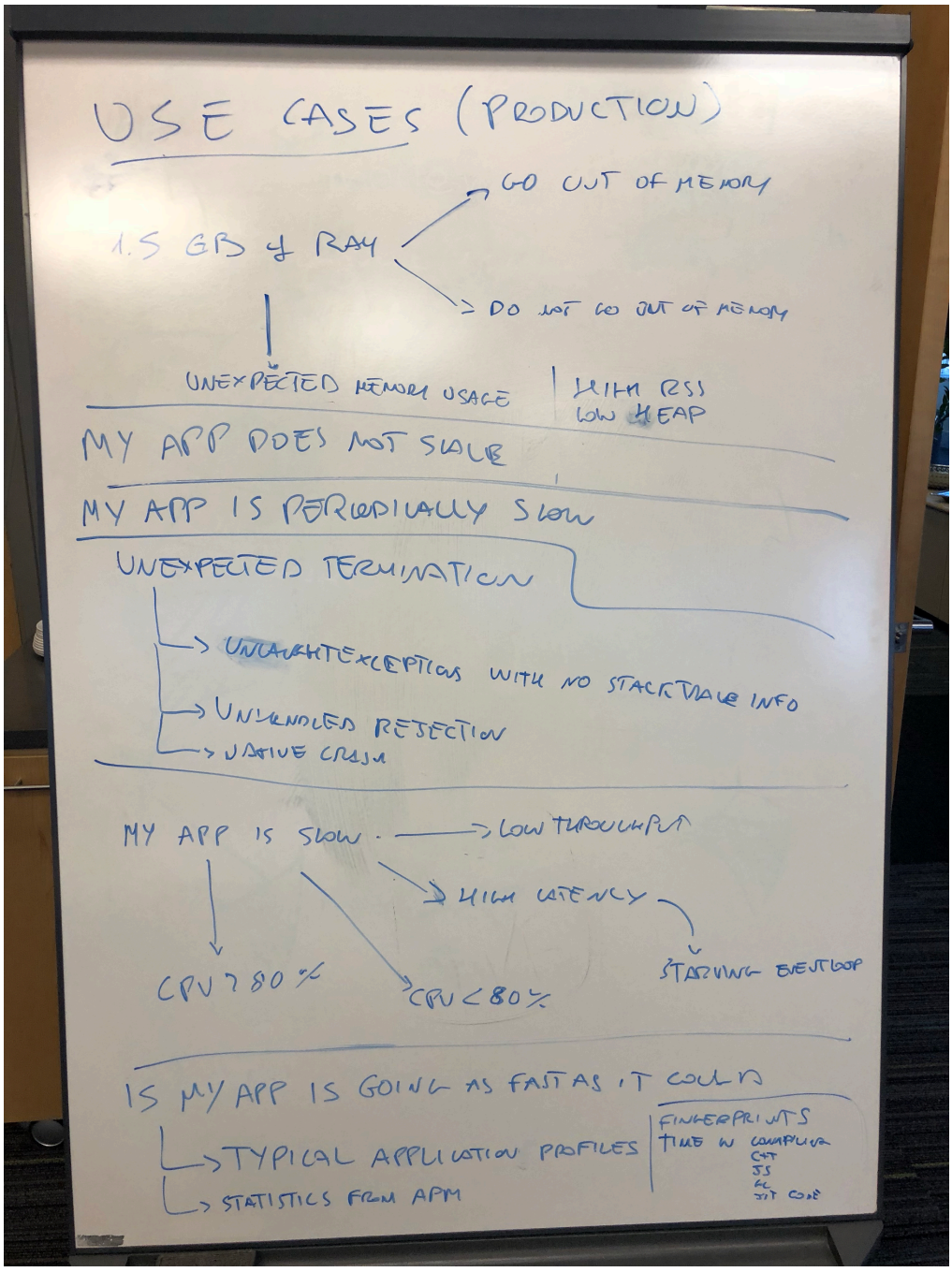
Platform neutrality/APIs

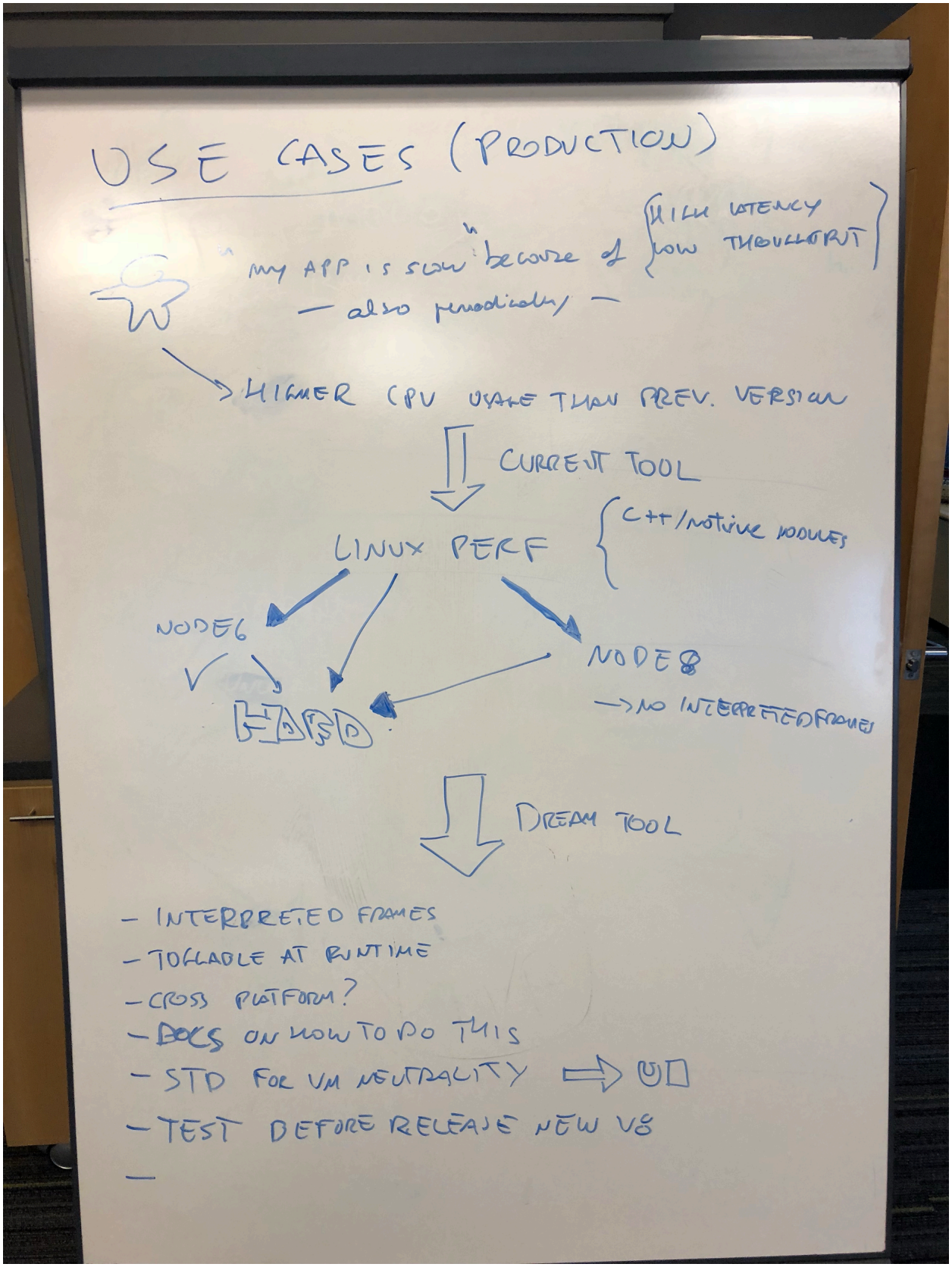
1. Platform neutrality/APIs
2. Expanding N-API to cover diagnostics APIs ?
3. Platform neutral internal API
4. What is the VM API contract with Node ?
5. Standardize/convention about a tracing mechanism in core dtrace vs lttng vs trace_event, ….
6. Current Trace tools compatibility table across platforms
7. What probe points do we really need
8. How can we stay away from platform specific tooling
9. Standardized hooks to get out of APM Monkey patching of modules

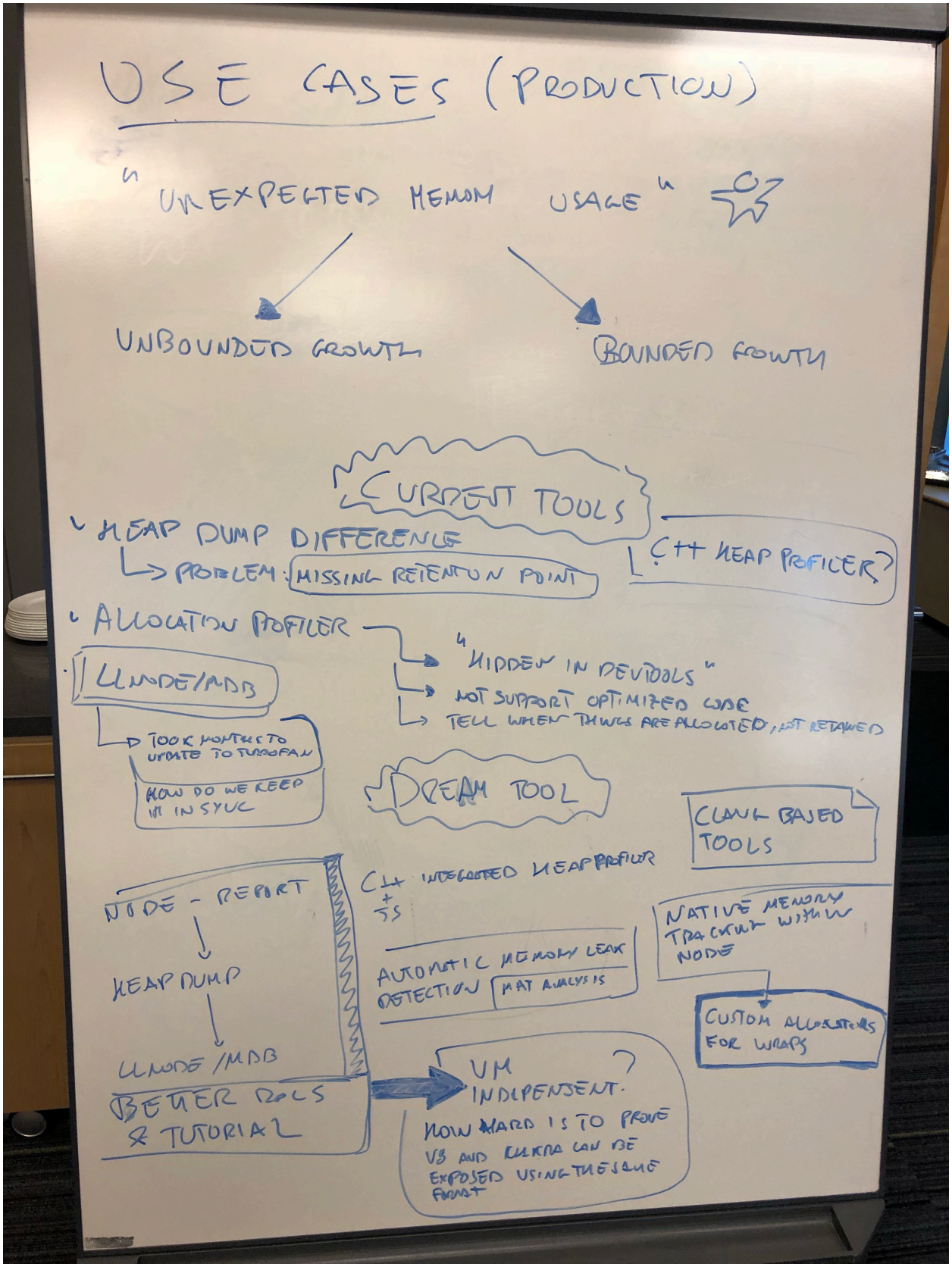
What’s missing?

- How does Node.js stack up against other languages in terms of diagnostics
- Would be good to be able to capture what we have, what we need to work on and have deck to position that Node.js is good from a diagnostics point of view (possibly after some work)

Use cases and tools




USE CASES (PRODUCTION)
MY APP IS SLOW
— also periodically —
because of
HIGH LATENCY
LOW THROUGHPUT
HIGHER CPU USAGE THAN PREV. VERSION
CURRENT TOOL
LINUX PERF
C++/NATIVE MODULES
NODE6
NODE8
→ NO INTERPRETED FRAMES
HARD
DREAM TOOL
- INTERPRETED FRAMES
- TOGGLABLE AT RUNTIME
- CROSS PLATFORM?
- DOCS ON HOW TO DO THIS
- STD FOR VM NEUTRALITY
- TEST BEFORE RELEASE NEW V8
-


USE CASES (PRODUCTION)
"UNEXPECTED MEMORY USAGE"
UNBOUNDED GROWTH
BOUNDED GROWTH
CURRENT TOOLS
HEAP DUMP DIFFERENCE
PROBLEM: MISSING RETENTION POINT
C++ HEAP PROFILER?
ALLOCATION PROFILER
"HIDDEN IN DEVTOOLS"
NOT SUPPORT OPTIMIZED CODE
TELL WHEN THINGS ARE ALLOCATED, NOT RETAINED
LLNODE/MDB
TOOK MONTHS TO UPDATE TO TURBOFAN
HOW DO WE KEEP IN SYNC
DREAM TOOL
CLANK BASED TOOLS
NODE - REPORT
HEAP DUMP
LLNODE/MDB
BETTER DOCS & TUTORIAL
C++ INTEGRATED HEAP PROFILER
+
JS
AUTOMATIC MEMORY LEAK DETECTION
MAT ANALYSIS
NATIVE MEMORY TRACKING WITH NODE
CUSTOM ALLOCATORS FOR WRAPS
VM INDEPENDENT.
HOW HARD IS TO PROVE V8 AND CHAKRA CAN BE EXPOSED USING THE SAME FORMAT