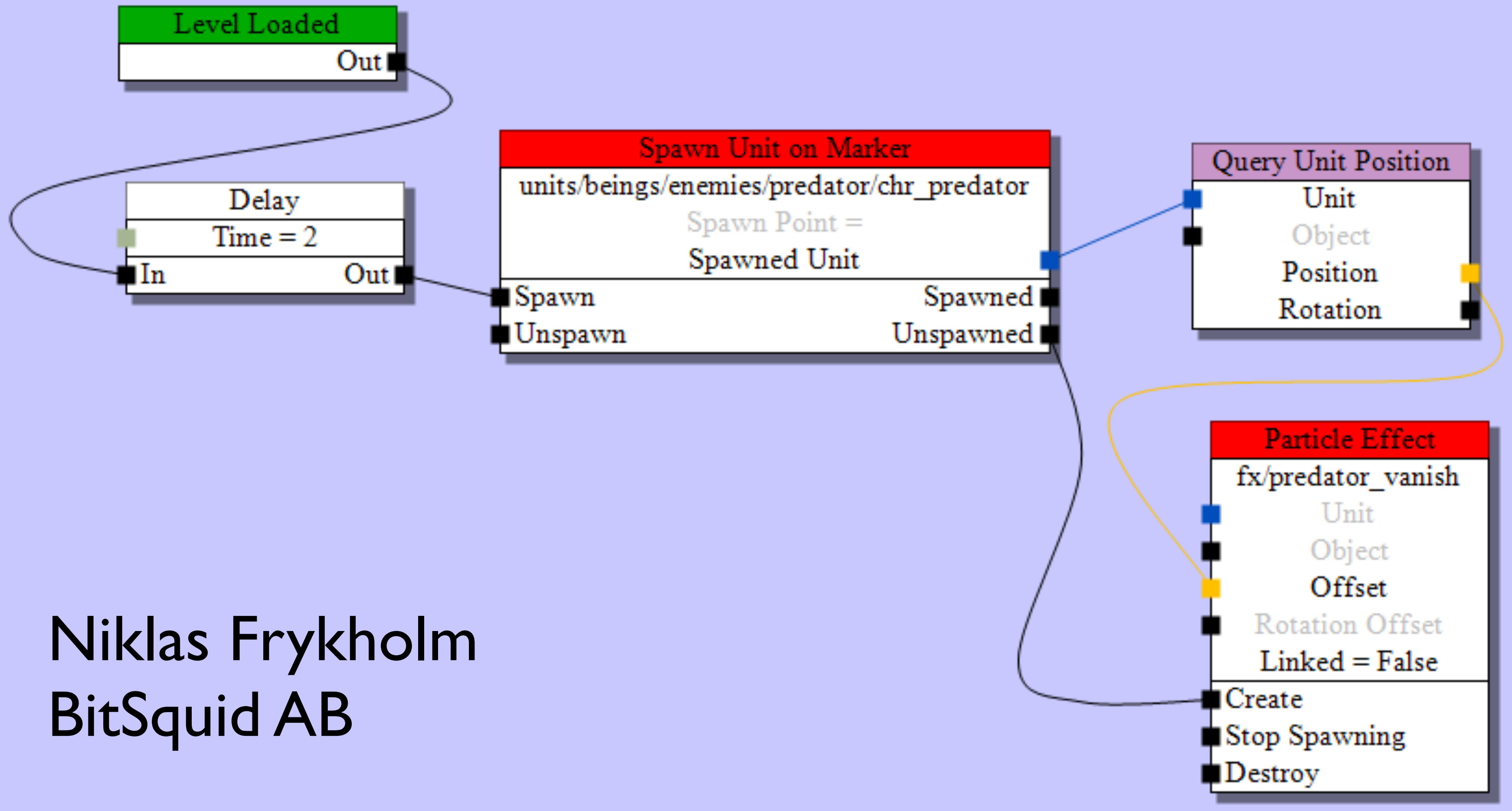
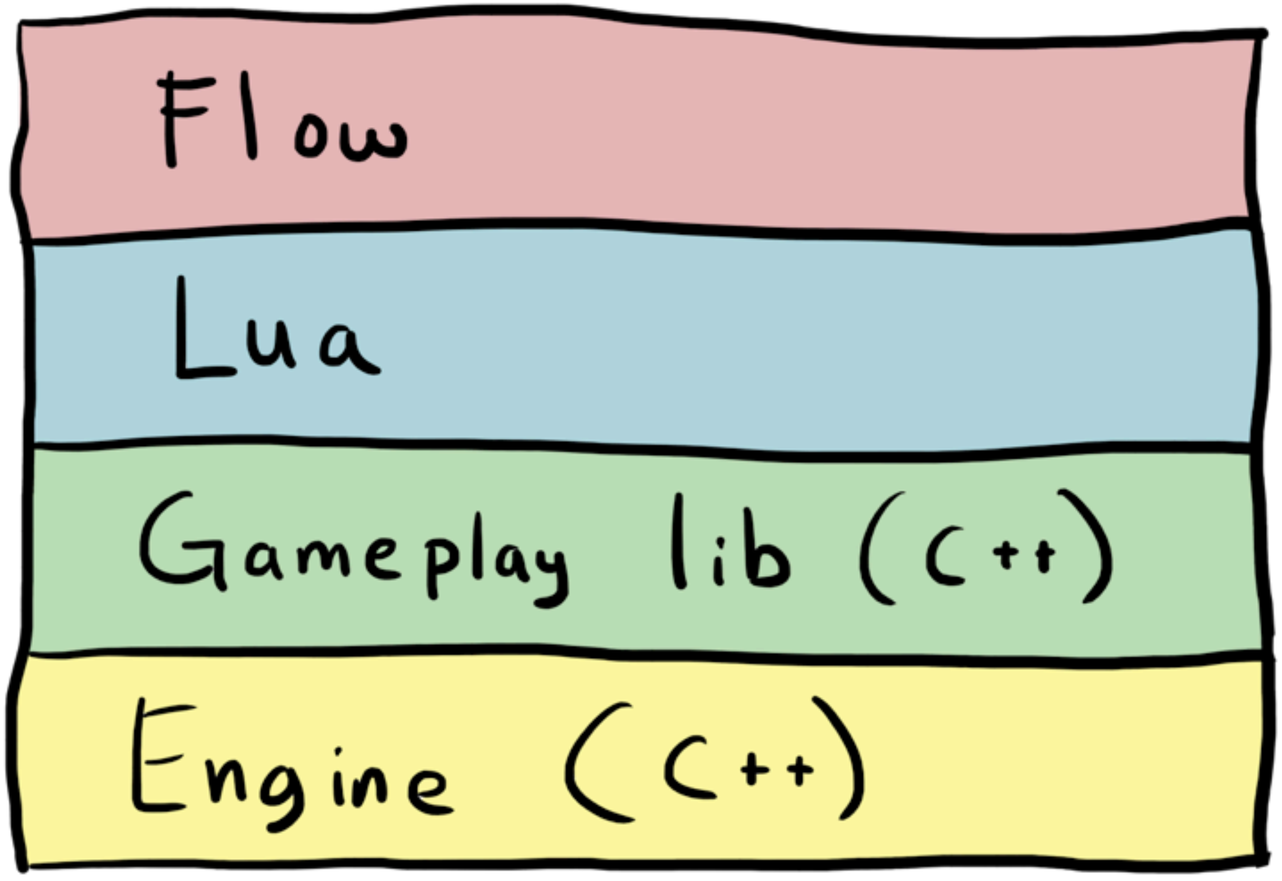
Flow

Data-Oriented Implementation of a Visual Scripting Language



Niklas Frykholm
BitSquid AB

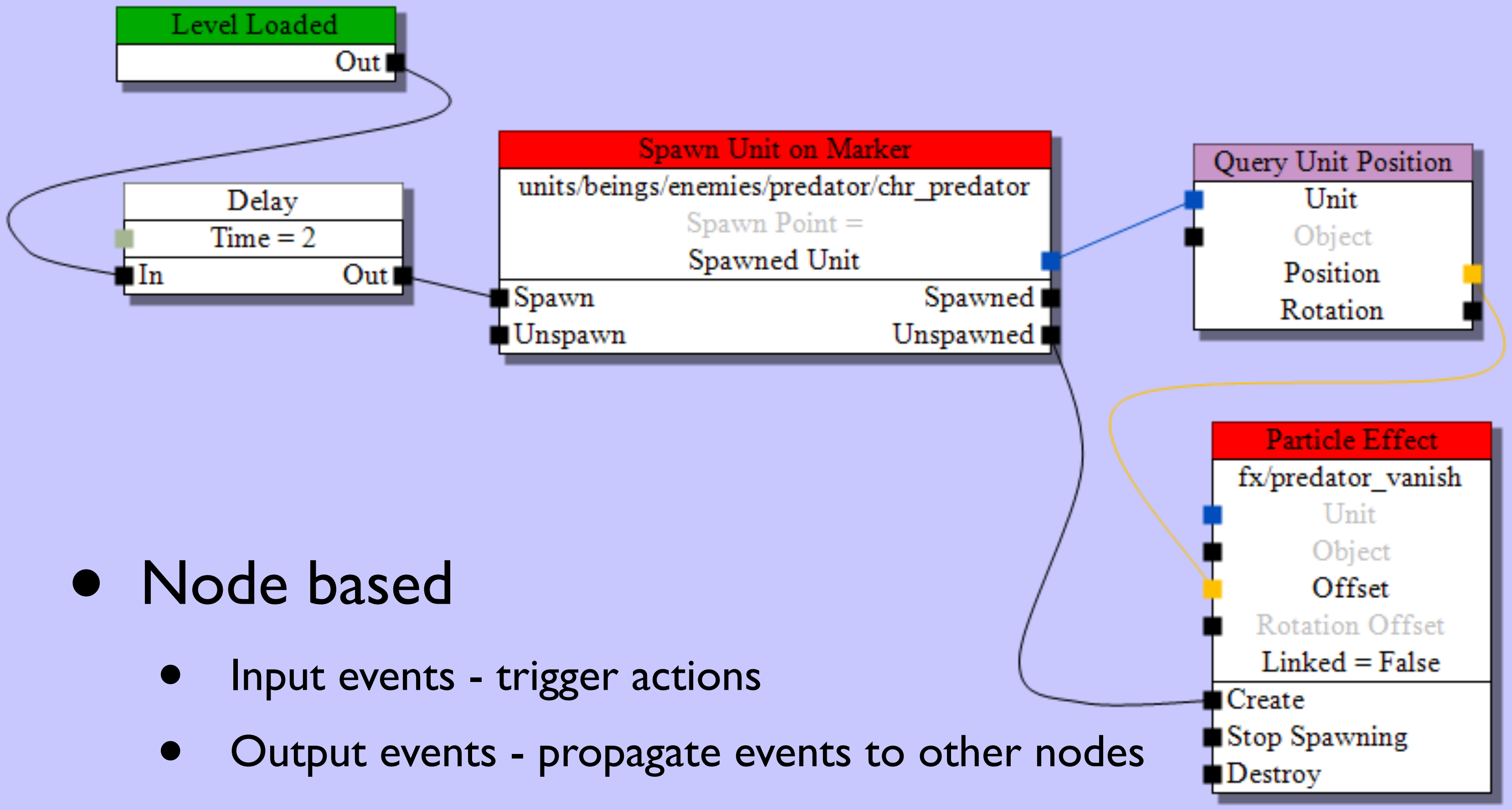
Levels of programming



Visual scripting benefits

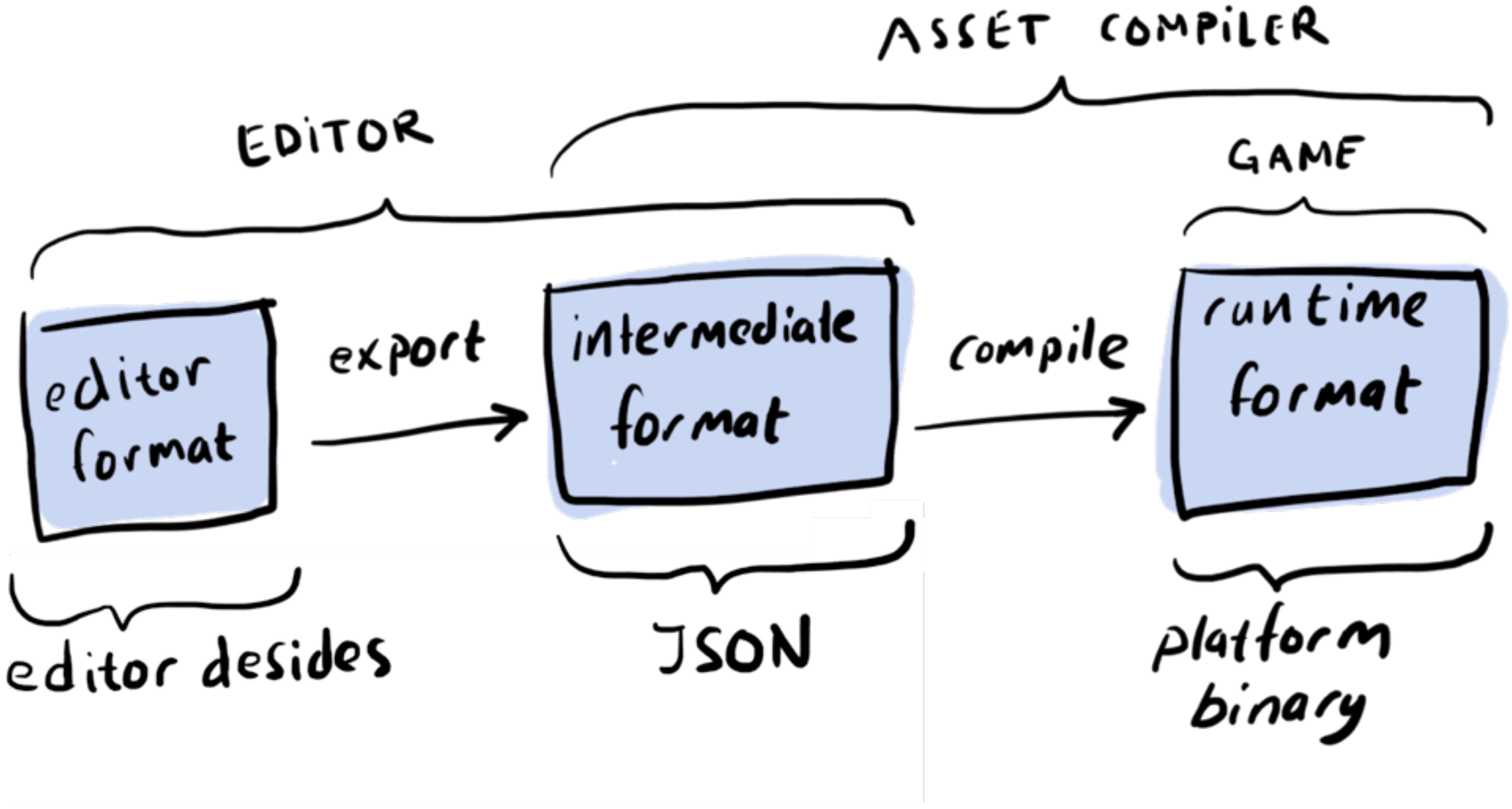
- To content creators
    - More power: integrate effects, sounds, etc
    - Immediately see results in-game: experiment
- To gameplay programmers
    - Less messy ”special purpose” gameplay code
- Performance
    - Faster than Lua (no VM, no GC, typed)

How Flow works



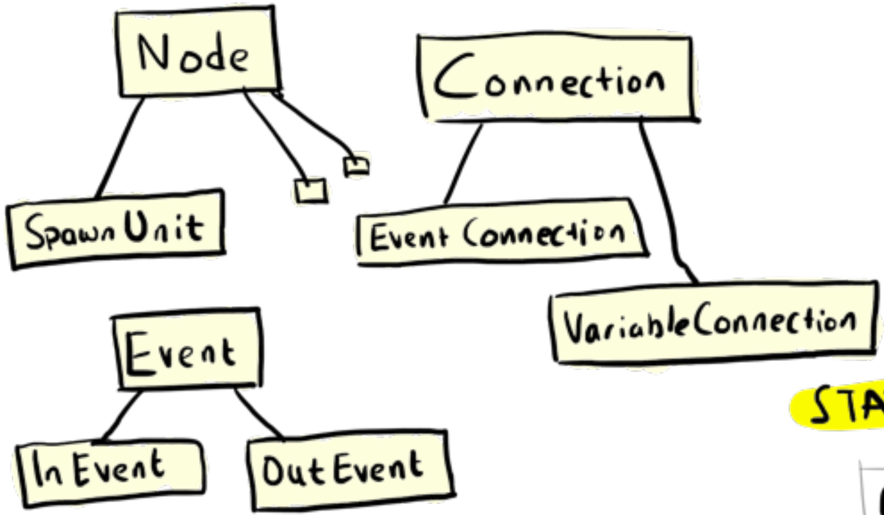
- Node based
  - Input events - trigger actions
  - Output events - propagate events to other nodes
  - Variables - shared data
- Inert - pay for what you use

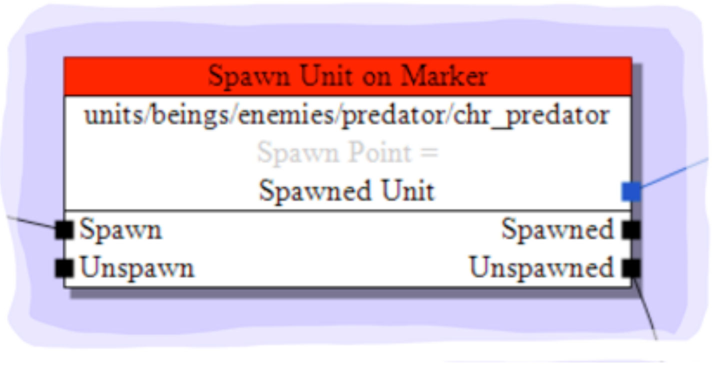
Flow in editor and runtime

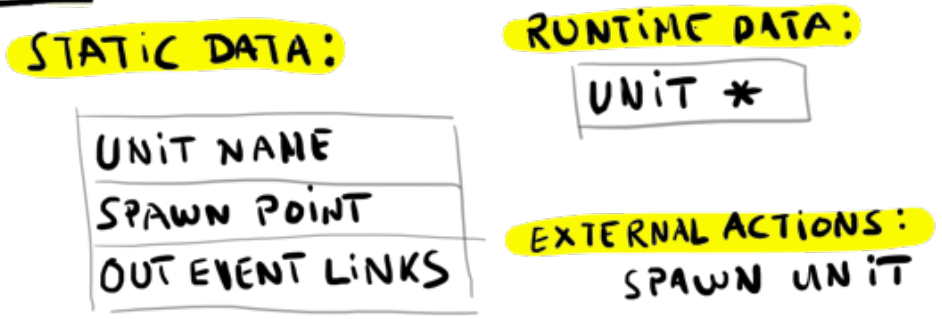


Data-oriented design

- Instead of thinking about classes
  - Think about data layouts and transforms
  - Actions on real-world items (bits and bytes) not relations between abstract concepts (classes and objects)

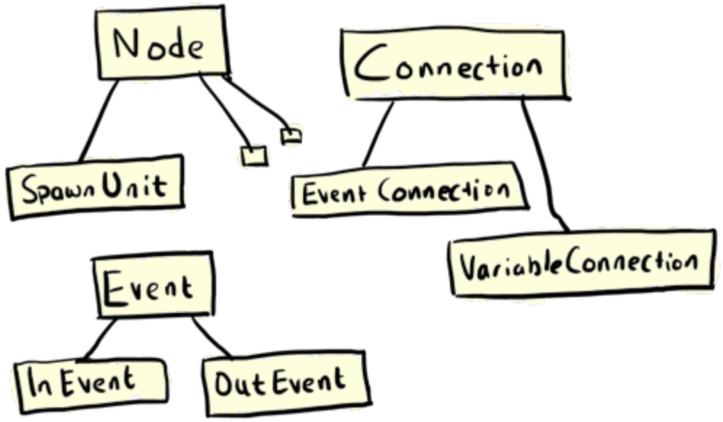






Motivation for DoD

- Performance
  - Memory is slower than CPU, think about caches
  - Focus on what the computer *actually does*
- ”Objects” do not always give a good design
  - Does EventConnection : Connection make sense?
  - Unnecessary abstractions
- Less coupling
- More freedom



”More freedom”

- Say you want to store data about a network game
  - String keys and values
- One possible implementation:

name\0Fight!\0map\0canyon\0players\020\0\0

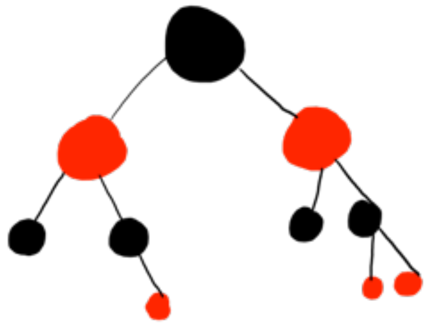
- Not OOD because ”there are no objects”
  - But this *might* be a good solution (depending on use)
  - DoD gives a wider design space

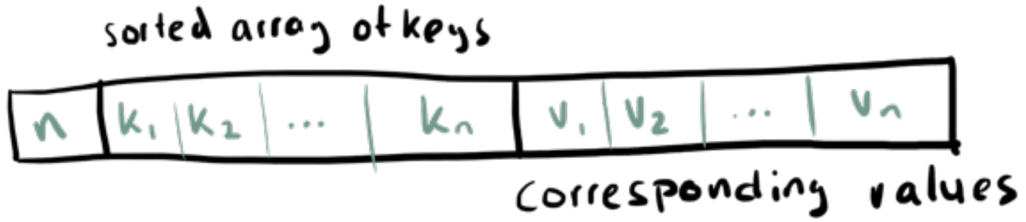
Static and dynamic data

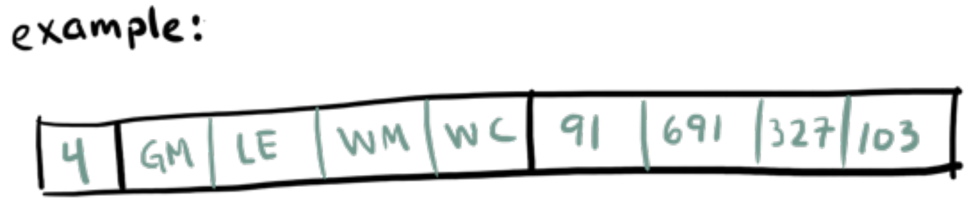
- Very different requirements
  - Dynamic data must change, grow and shrink
  - Dynamic data must (often) allocate memory
  - Dynamic data must be thread-safe
- Standard data structures are made for dynamic data
  - std::vector, std::map, std::string
- *Most* game data is static
  - Textures, animations, vertex data, etc
- DO NOT USE STANDARD DATA STRUCTURES FOR STATIC DATA!

Dynamic/static example

- std::map (red-black tree)
  - Lots of pointers, cache unfriendly
  - To make fast *modification* possible
- Sorted array
  - Find item with binary search
  - Simple, cache friendly, single memory allocation

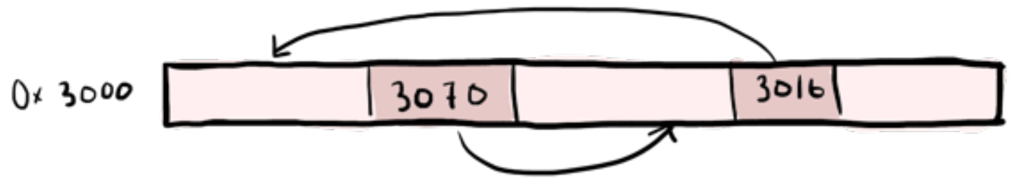




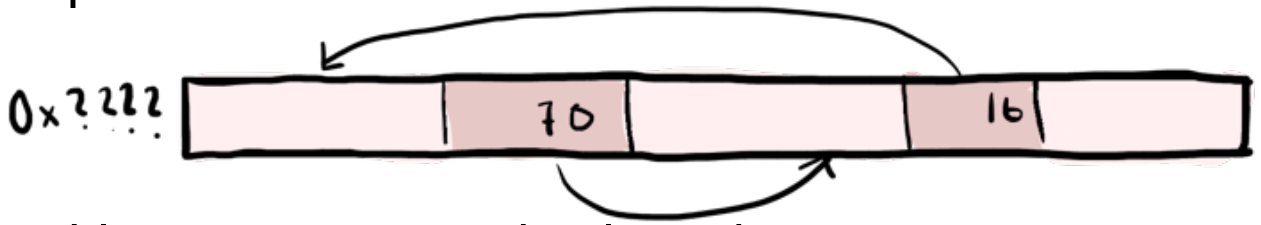


Data ”blobs”

- Since static data doesn’t need to grow we can always store it in a single memory block



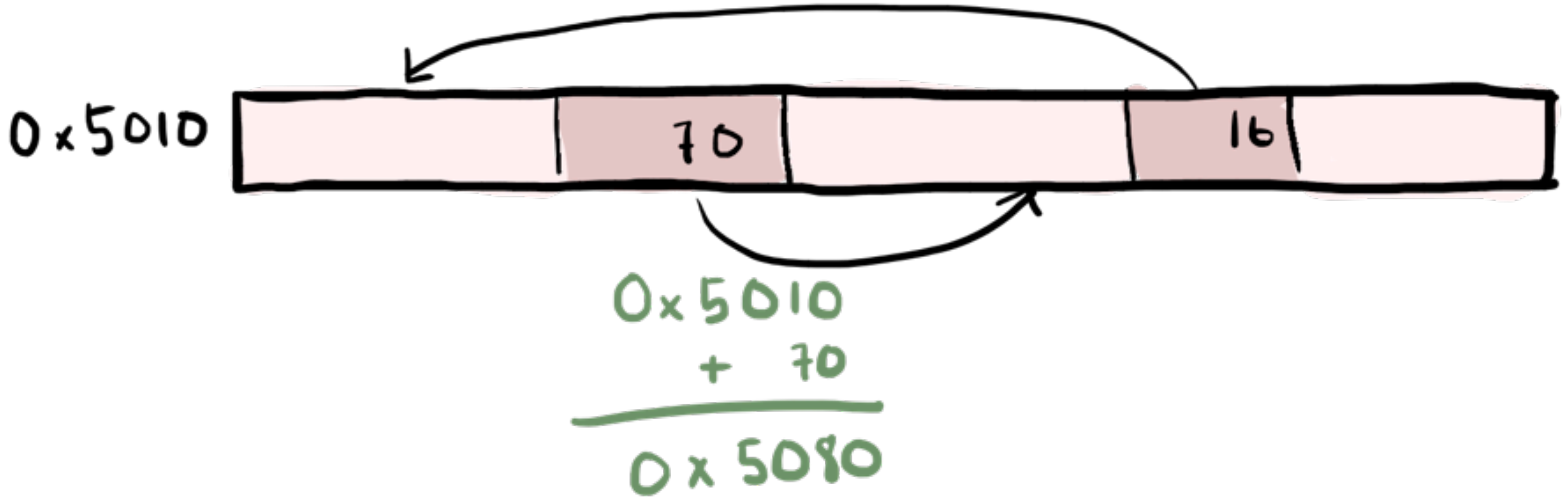
- Idea: make the data relocatable by using offsets instead of pointers



- Now we can move the data wherever we want
  - And read it directly of disk
- I call this ”The Blob”

Blob disadvantages

- Slightly more inconvenient
  - You must compute pointers from offsets



Blob advantages

- Automatically cache friendly
  - All data allocated at the same place
- Automatically memory system friendly
  - Few large allocations are better than many small
  - All allocation sizes known up front
  - No fragmentation during data load
  - Can be moved for defragmentation
- Automatically loads fast
  - Copy data directly from disk to memory
  - No pointer patching or other data fixup

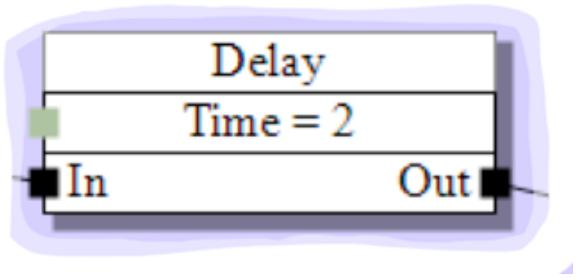
Blob advantages 2

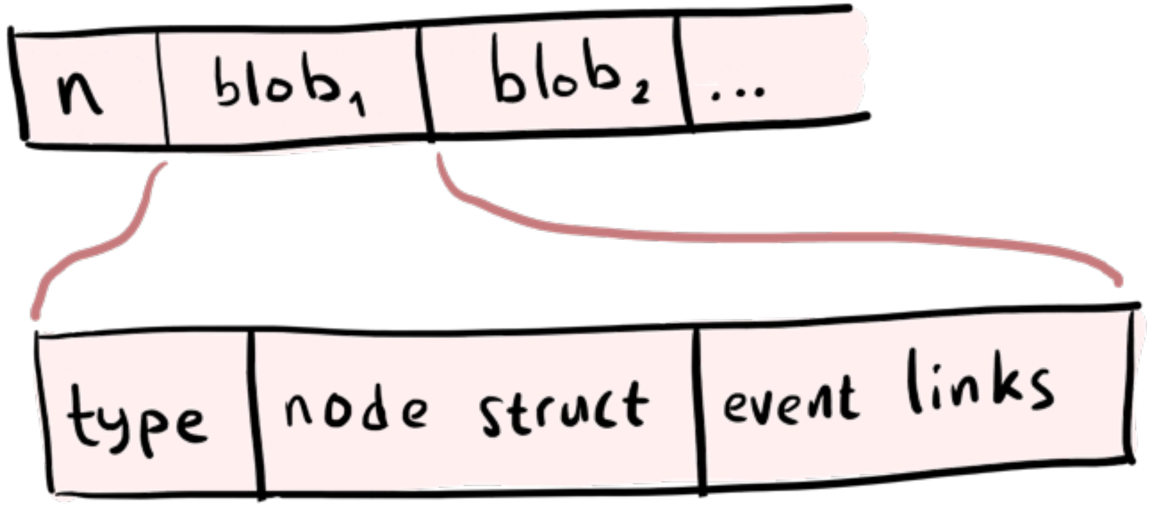
- Less work
  - Don’t have to think about serialization format
  - No need to write serializer/deserializer
- Data can be copied
  - To SPU for off-core processing
  - For instancing
- I ❤ the blob
  - And you should too!

Using blobs for Flow

- struct for each node's data

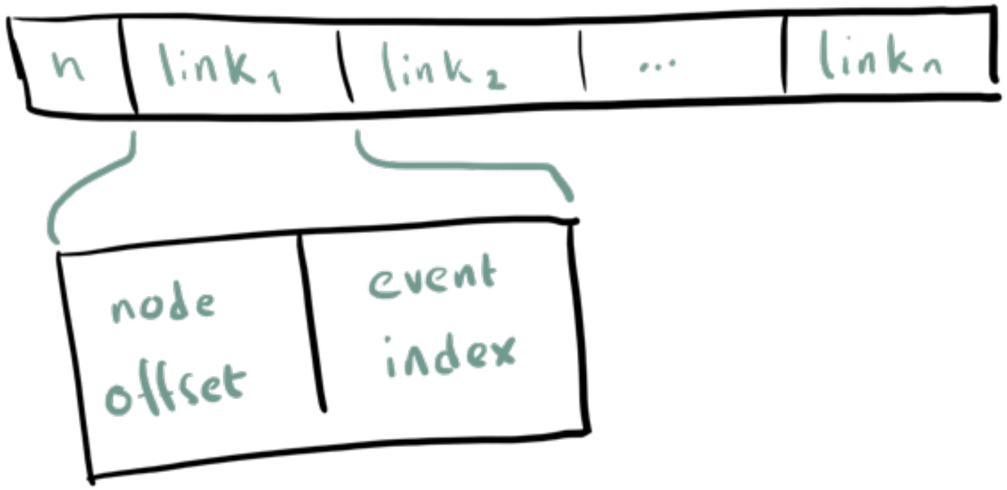
```
struct DelayNode {
    float time;
};
```

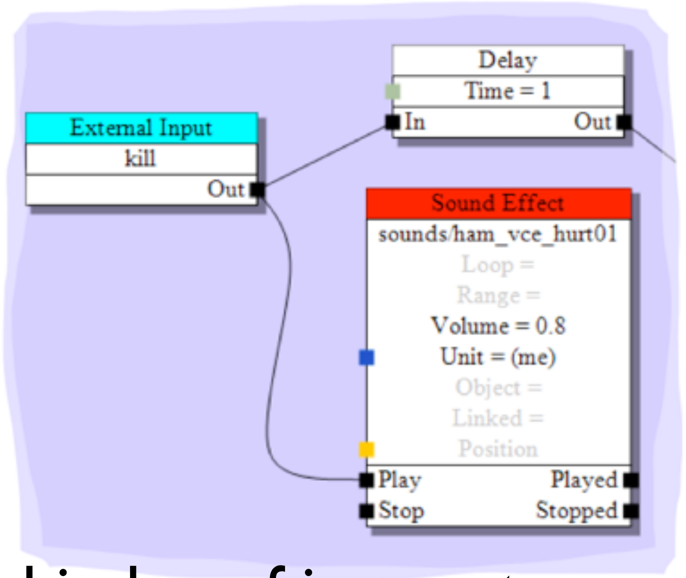




Storing the event links

- For each out event, store linked events

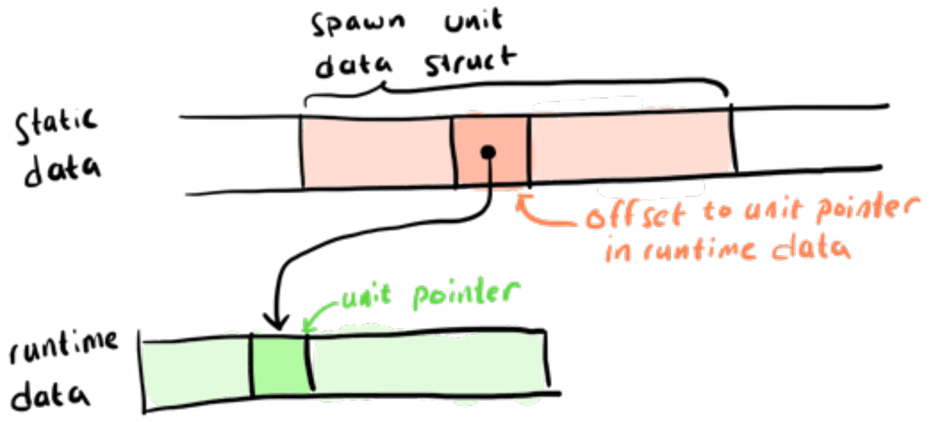


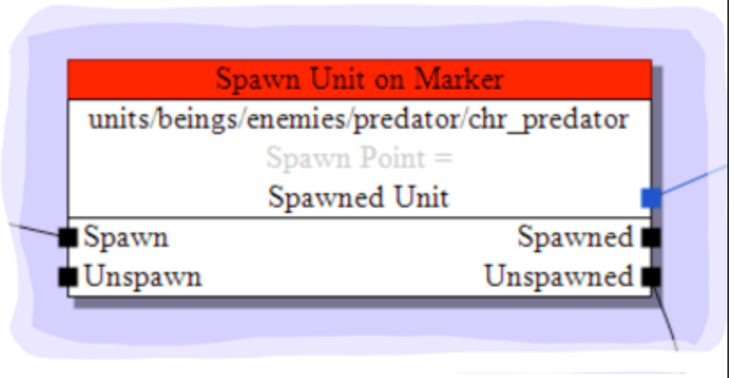


- Link: offset of target node and index of in event
  - Can loop over links to trigger an impulse

Runtime data

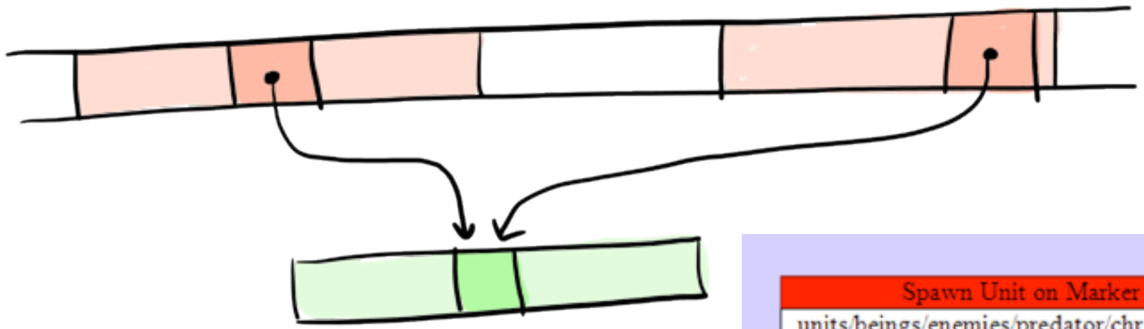
- Each node needs to store runtime data
  - Id of instances
  - Data on wires
- Idea: Runtime data is non-const blob
  - During asset compile nodes reserve memory
  - Get an offset into the runtime data

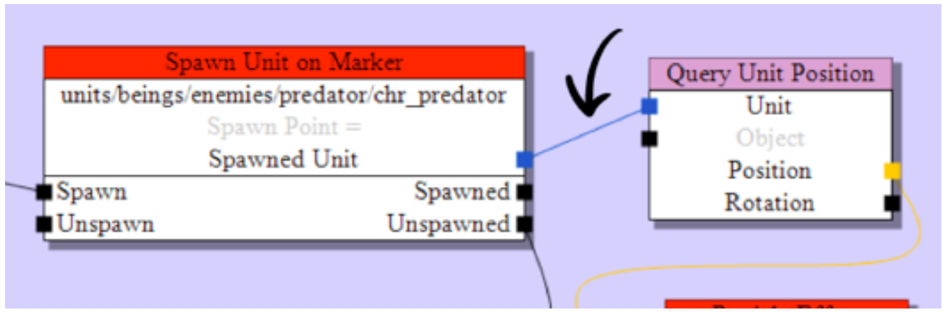




Data connections

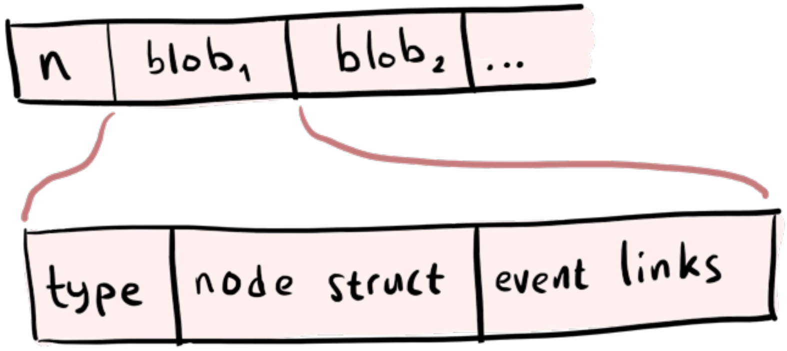
- Represent shared data
- Implementation: Share runtime data
  - Two offsets pointing to the same location

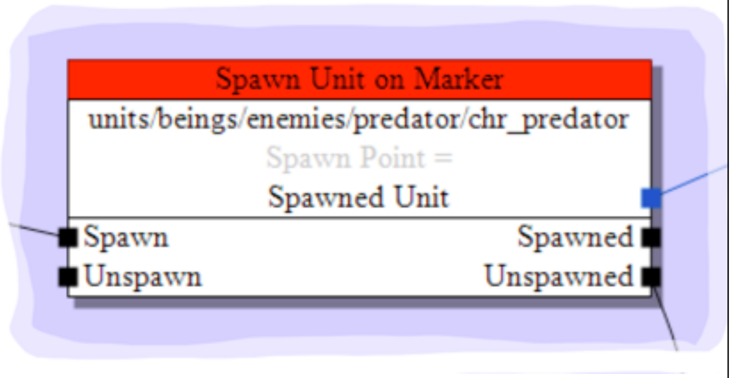




Running a node

- Node actions implemented by functions
  - void spawn_unit(const SpawnUnit *u, int event, char *runtime_data)
- Lookup function pointer based on type
  - node_functions[node_type](blob, event, runtime_data)
  - Function table acts as vtable

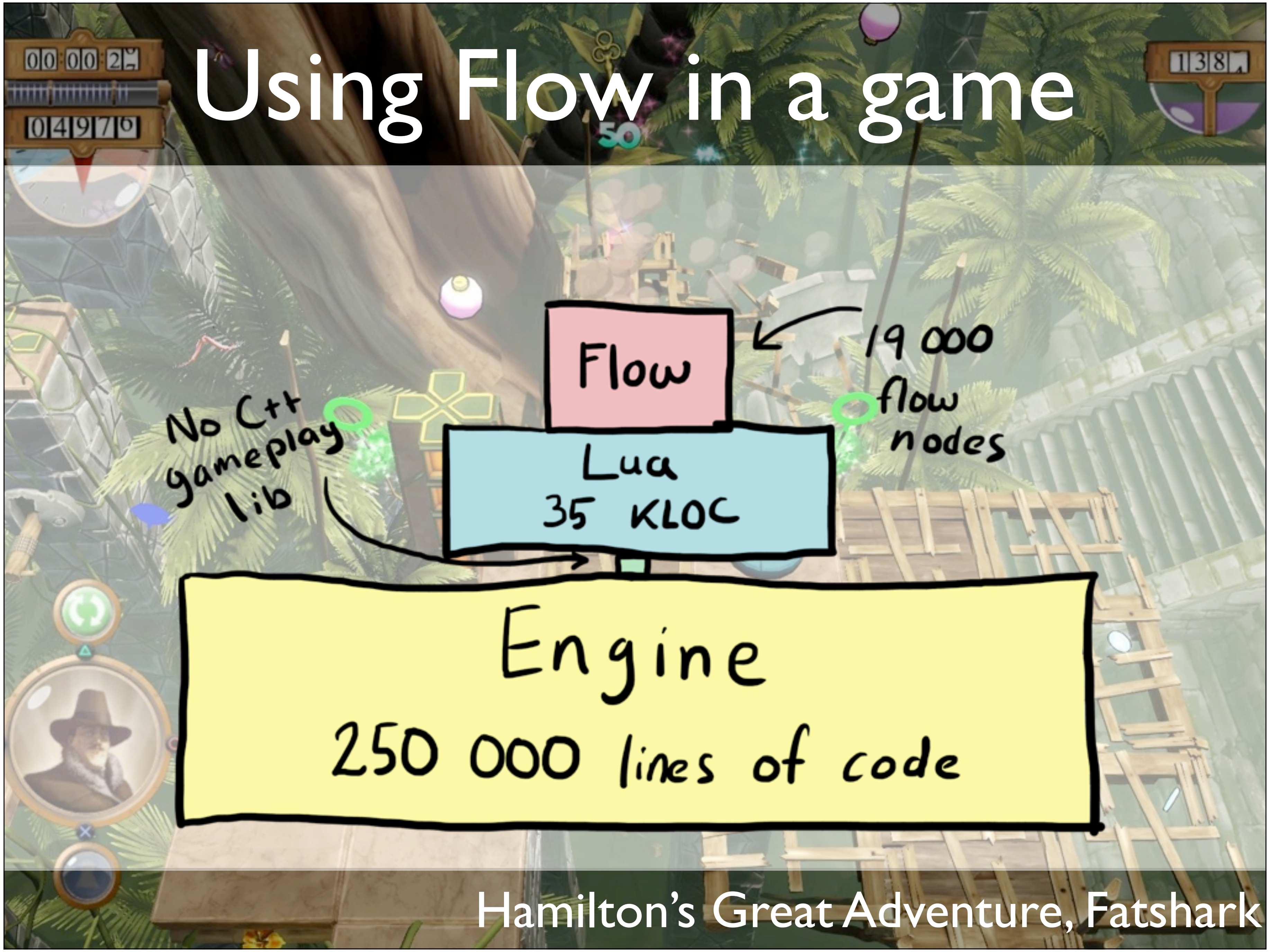




Using Flow in a game

Hamilton's Great Adventure, Fatshark

Using Flow in a game

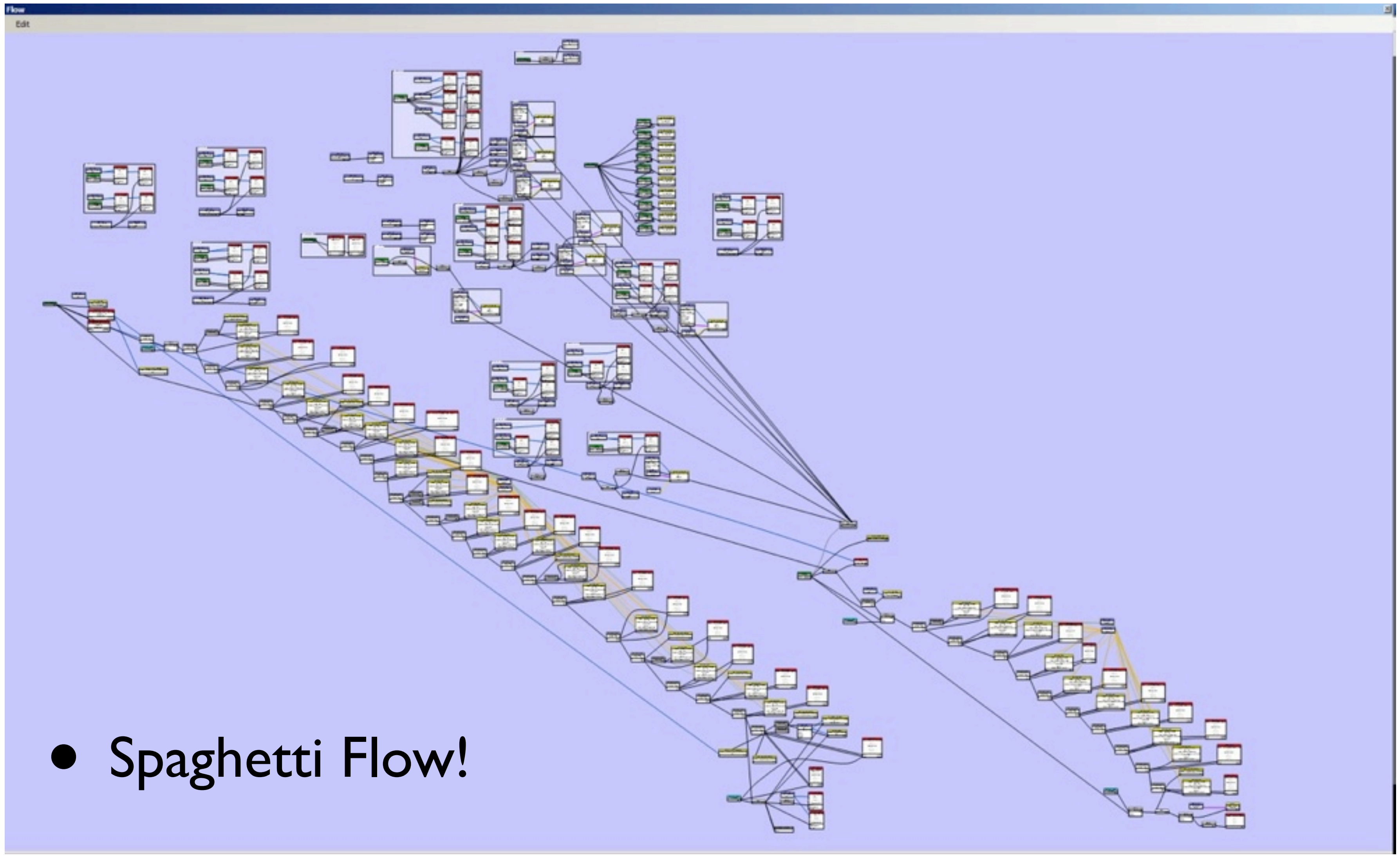


Hamilton's Great Adventure, Fatshark

The Good

- Able to do more and work faster
  - Easier than traditional scripting or ”mission XML files”
  - Add quality: delays, randomization, multiple effects, polish
  - Test ”crazy ideas” without code support
  - Everyone can experiment with ”behaviors” and ”rules”

The Bad & the Ugly



- Spaghetti Flow!

Q&A

- Questions?
  - Email: niklas.frykholm@bitsquid.se
  - Twitter: niklasfrykholm