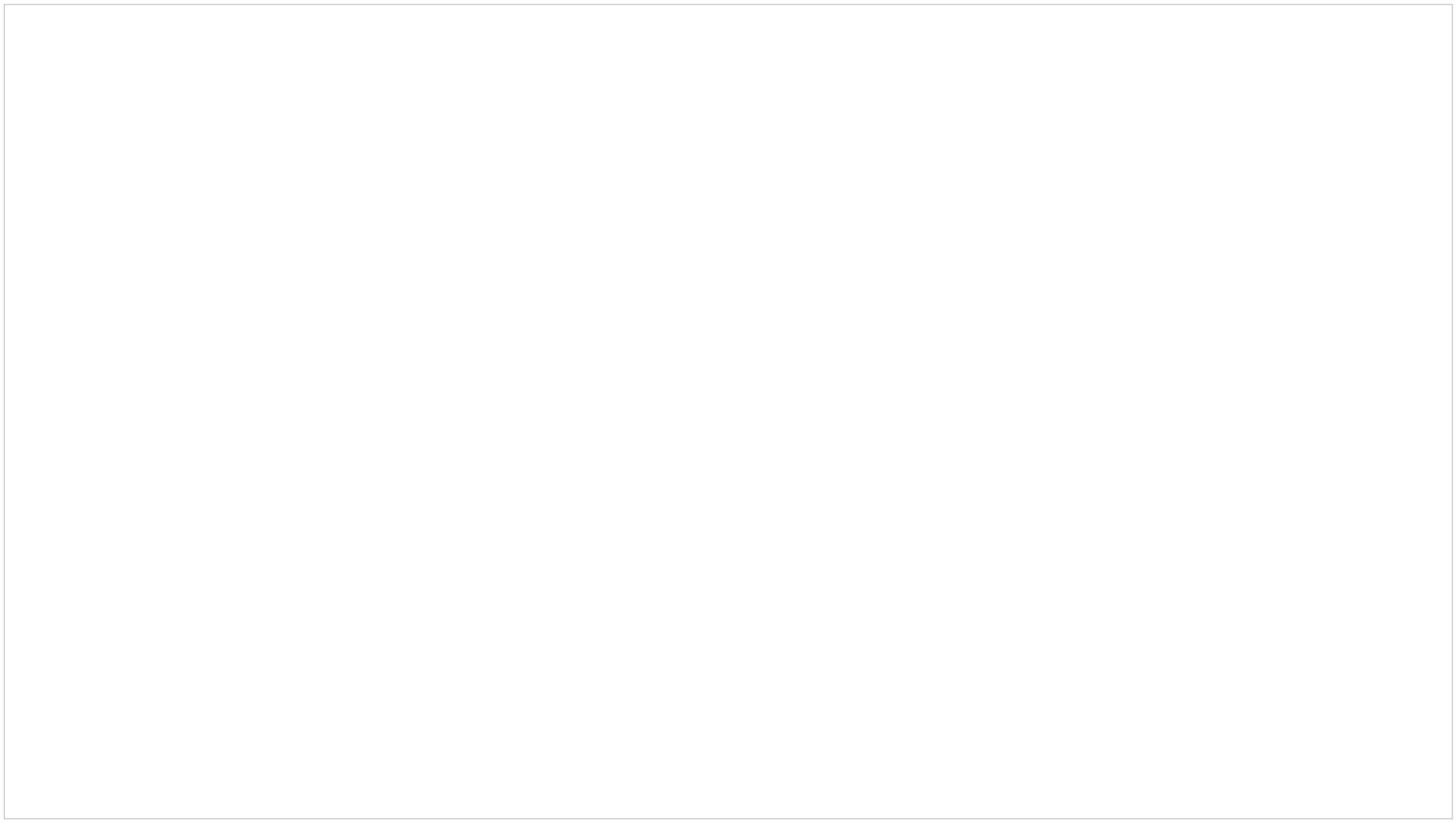
# Parsl Monitoring Message Flow



I’m going to talk about message flows in Parsl monitoring.

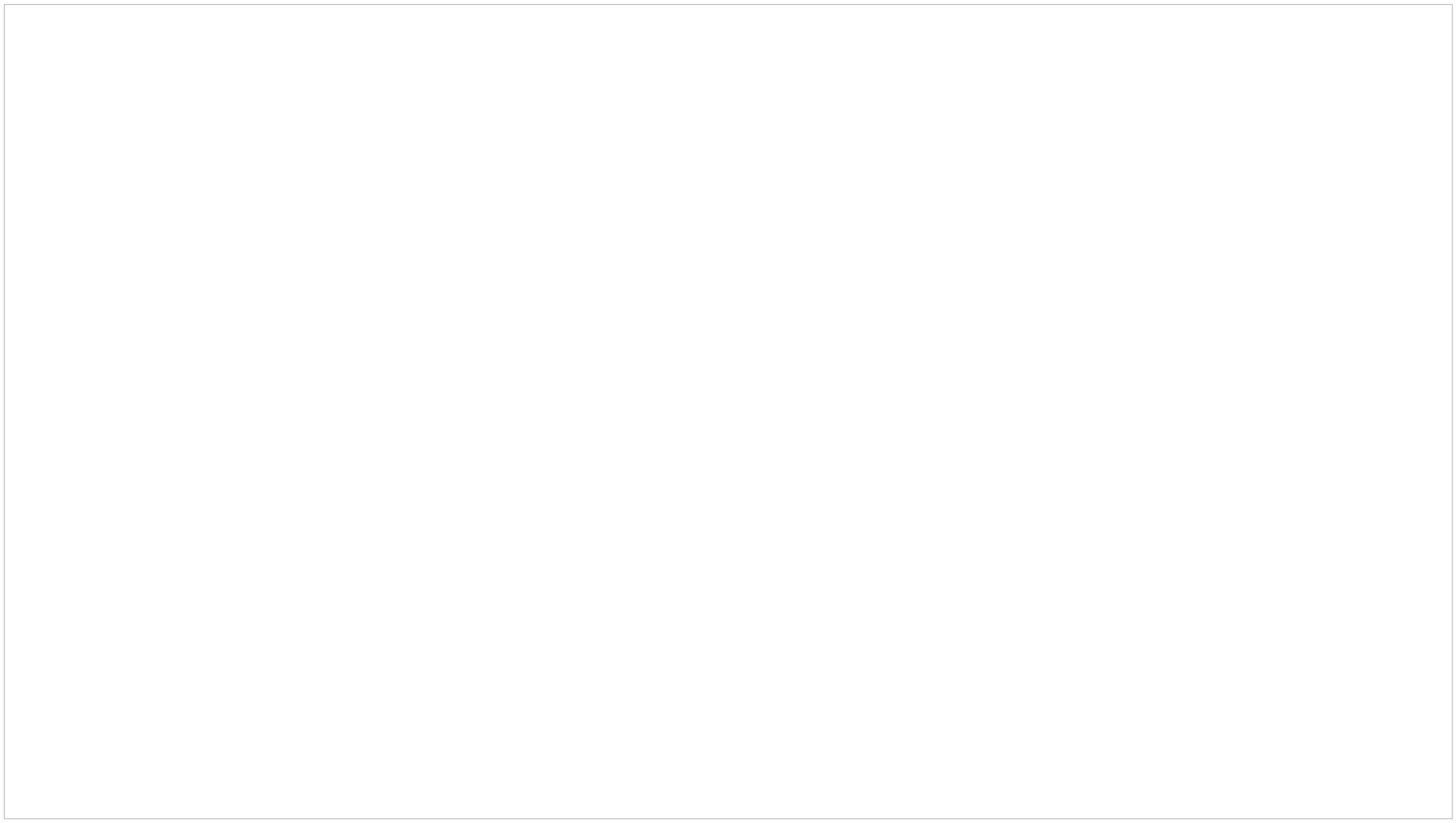
and what you can do with and do to those flows.

What monitoring messages look like is:
there are various sources of monitoring messages
(especially highlight which parts are also present in globus compute - to help GC devs)

talk about each message type, and its source(s)

they flow in different ways towards the MonitoringHub (and inside that an SQL-based database manager) and are stored in a relational model.

eg. table for tasks, table for tries, table for blocks of workers, ...



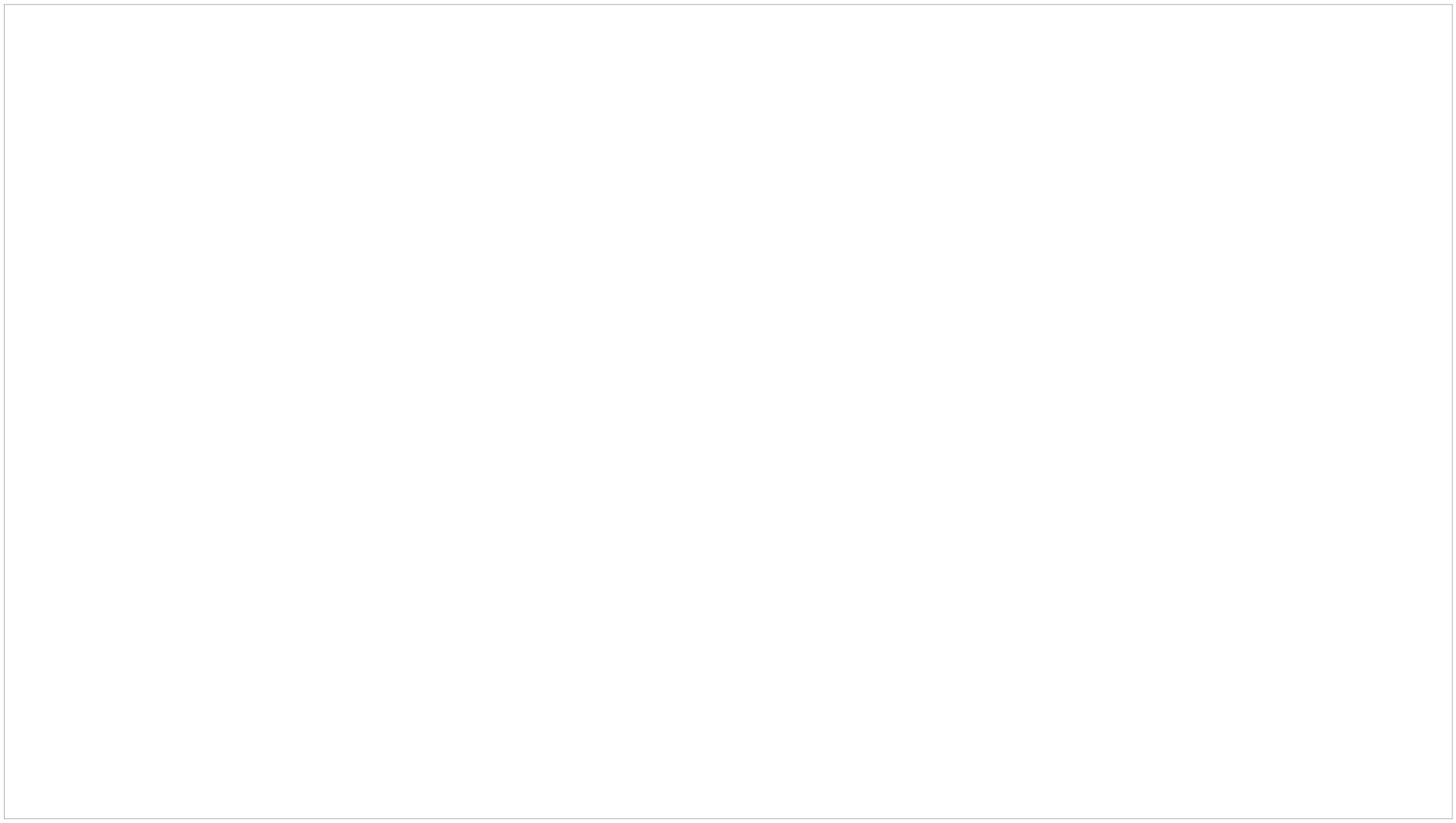
What I've been doing on monitoring in the last years is mostly rationalization and configurability

The different ways messages got to the monitoring hub was very ad-hoc. [diagram that represents how things were 1-2 years ago]

UDP, filesystem, ZMQ, multiprocessing queues

Over the years we've had people want to hack on this code - making Parsl more hackable has been a goal of mine, as part of community work. part of that is making code more modular and explainable, part of that is encouraging people to screw around with it.
two things there: rationalise code to make it more understandable. make APIs/user-accessible plugin points.

eg. i) want to take monitoring data from workers to the submit side by some new (eg. scalable) mechanism. eg. chronolog last year
ii) want to consume monitoring data in a different way - plenty of people read from SQL database, but what about a more real-time consumption?



Pluggable radios

pluggable radio model, more formalised:

there’s a submit side multiprocessing queue.

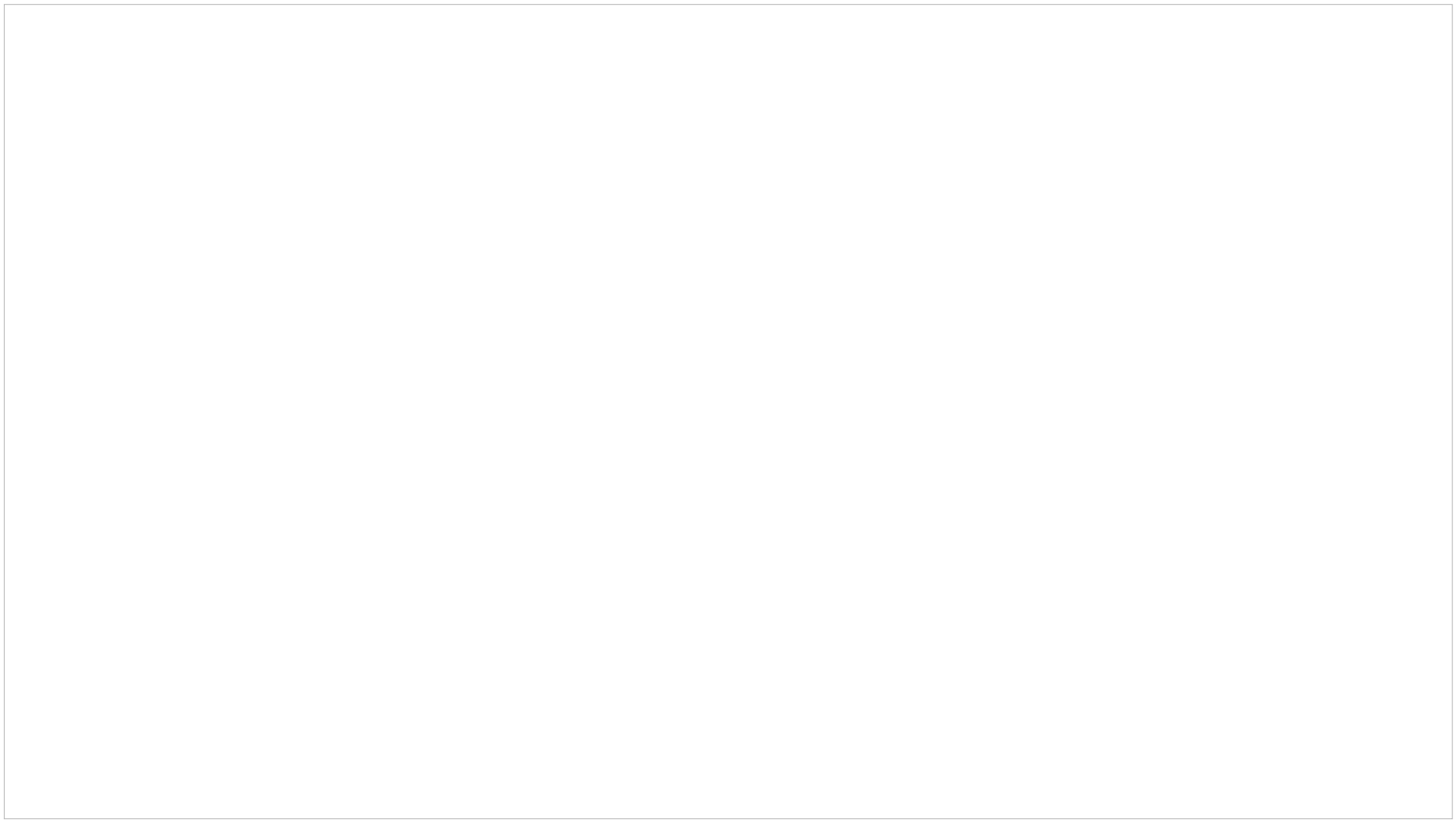
everything flows towards that queue - it’s the *only* way to get stuff to the monitoring hub/database manager.

database manager only listens to that queue

radio configuration / radio sender / radio receiver

interchange: ZMQ radio - htex starts a zmq radio receiver
submit side code: directly into queue
worker wrapper: user supplied radio pair

using WQ/TV: your choice is (by default filesystem) and then UDP. they’re both a bit icky, for different reasons - UDP unreliable (yes, i have experience this in real life Parsl, its not just something your networking professor said might happen). filesystem - quite heavy load on shared filesystem metadata.
but - this is the plugin point for experimentation with that.

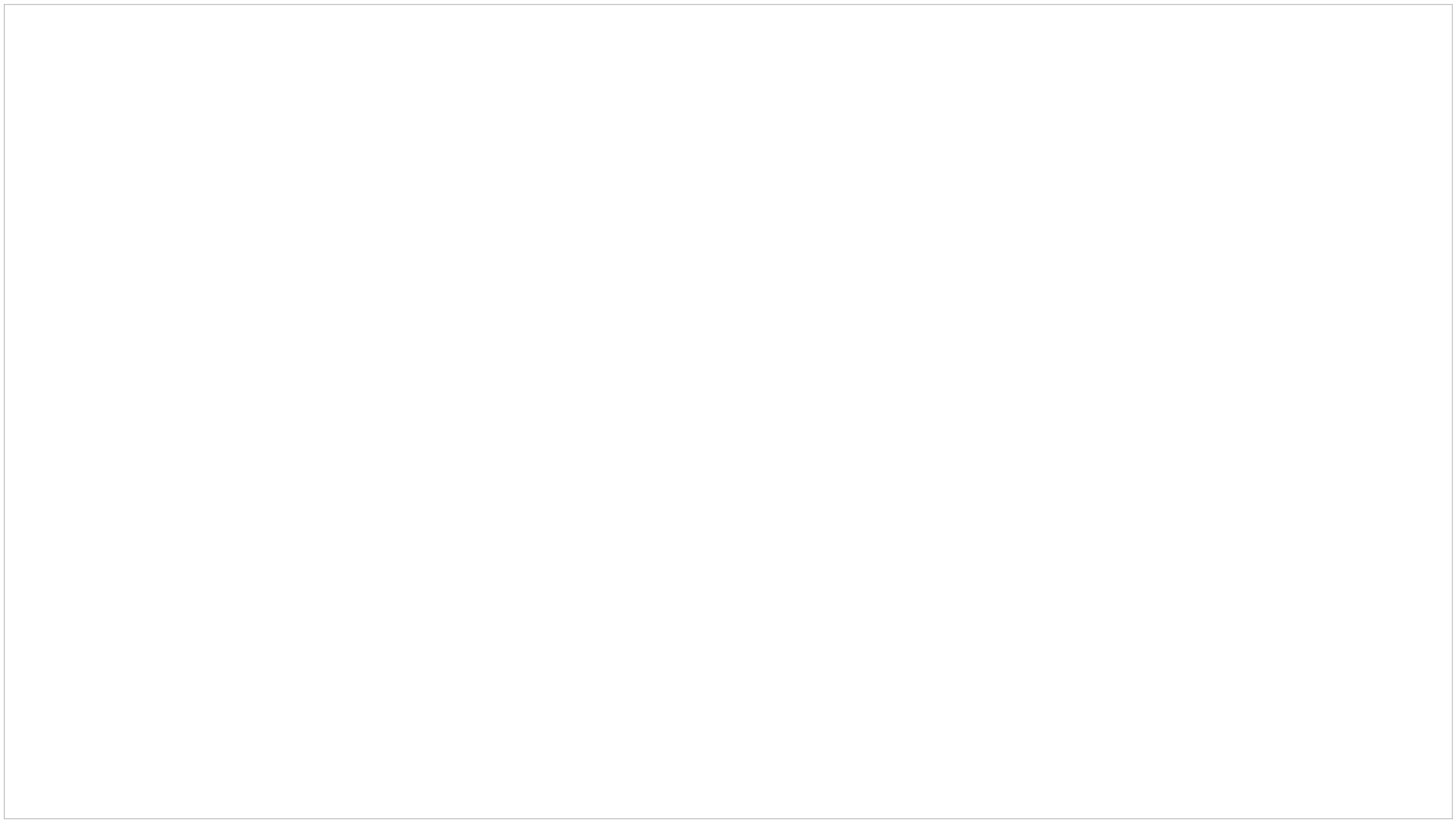


new radios / radio functionality

thread radio: thread sender can put stuff directly in the multiprocessing queue: there’s no need to go over any other protocol

UDP radio: configurability of hmac, to make this resilient against networking

htex - has a channel back to the submit side - thats how results come back! so (in year YYYY PR #NNNN) I added that in, also ad-hoc

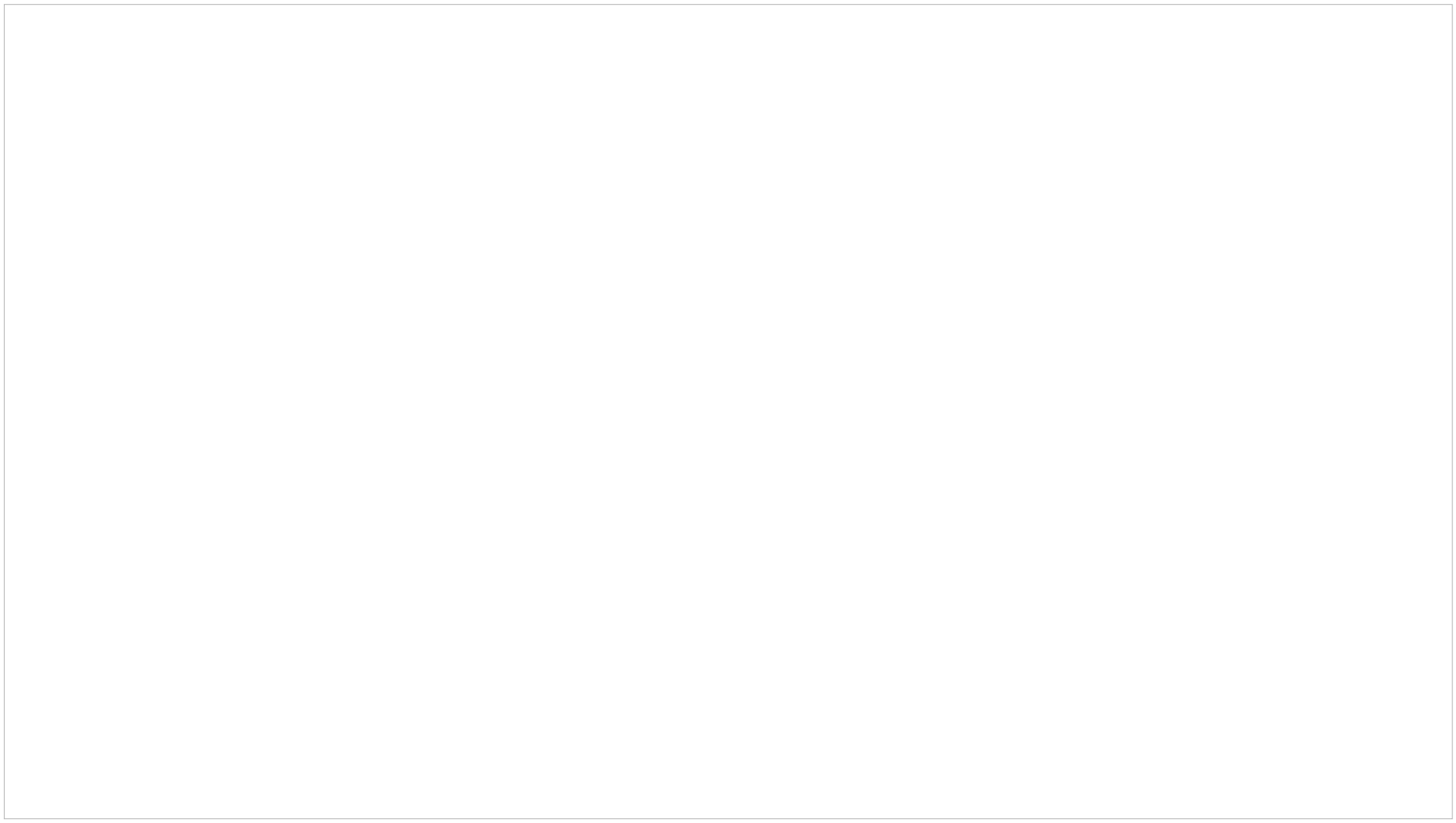


fewer radios

previously listeners for everything running whenever monitoring was turned on: no one was using UDP radio by default, but the UDP listener was always running, and in a tight polling loop!
bad things: CPU usage, network exposure

likewise filesystem radio...

if you’re using htex and local thread executor (for example, htex for real work, and then some join apps (which implicitly run in local thread pool)) - you don’t need *any* outwards facing network ports beyond what you have with monitoring turned off. or polling listener loops.



pluggable monitoring hub/database

pluggable *destination* for monitoring messages

this is not something I've done - but I pushed the architecture towards other people being able to experiment with this

hourglass model: we have this multiprocessing queue, and parsl coordinates all the monitoring messages going into this queue.

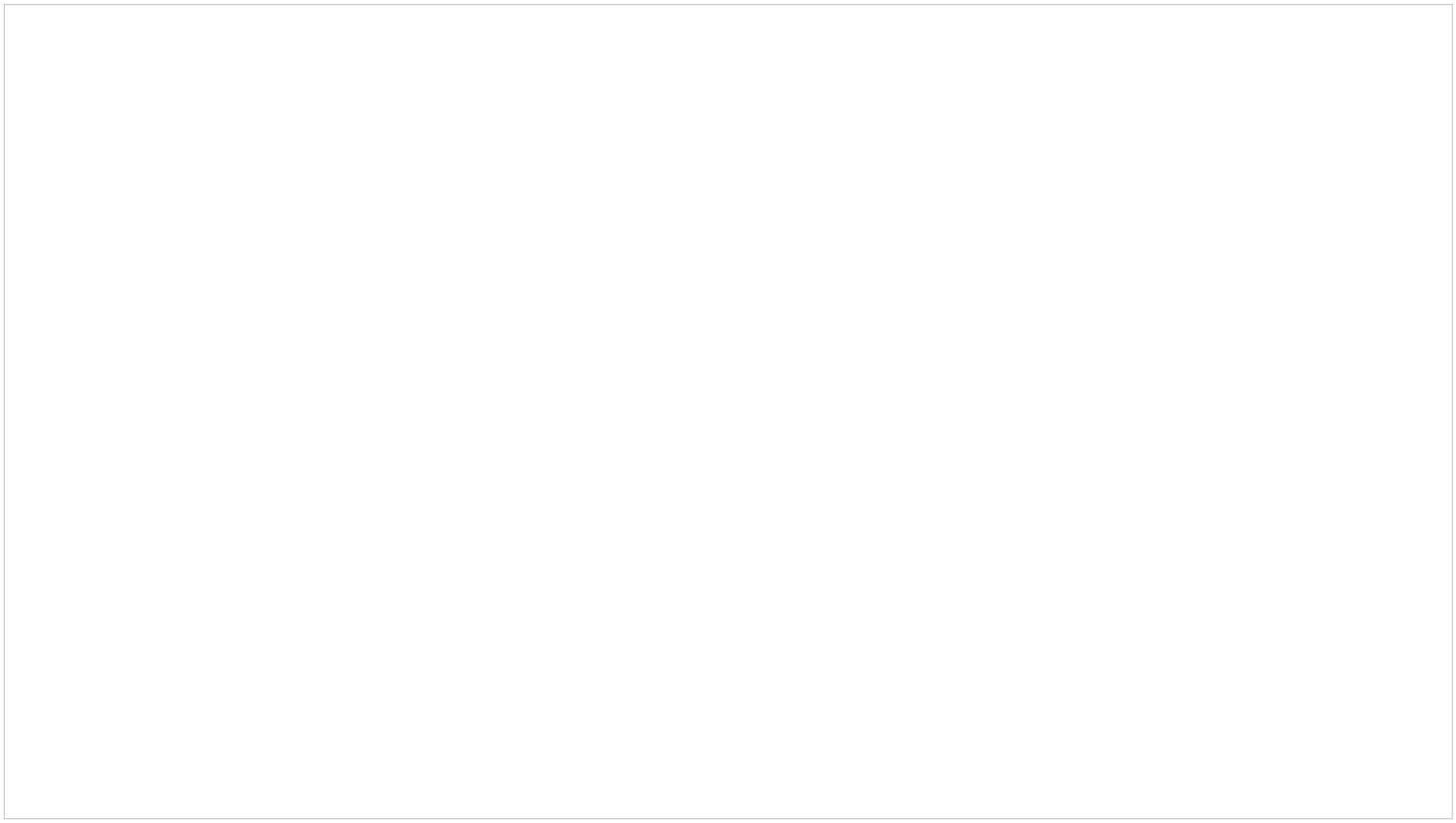
only the monitoringhub/database manager take stuff out of this queue - very modularised now.

anything else that can understand monitoring messages can fit there -- i encourage you to have a play.

examples:

if you're submitting into Parsl and a bit annoyed by accessing live data via SQL, grab the message stream instead?

if you've got a platform that wants to ingest data and can collect it all over the place - experiment with that: and in that case, you don't *have* to go via the message queue for remote radios, for example - different diagram for that.



overview

message flow diagram (as it is now)

with labelled plugin points + parsl-supplied options