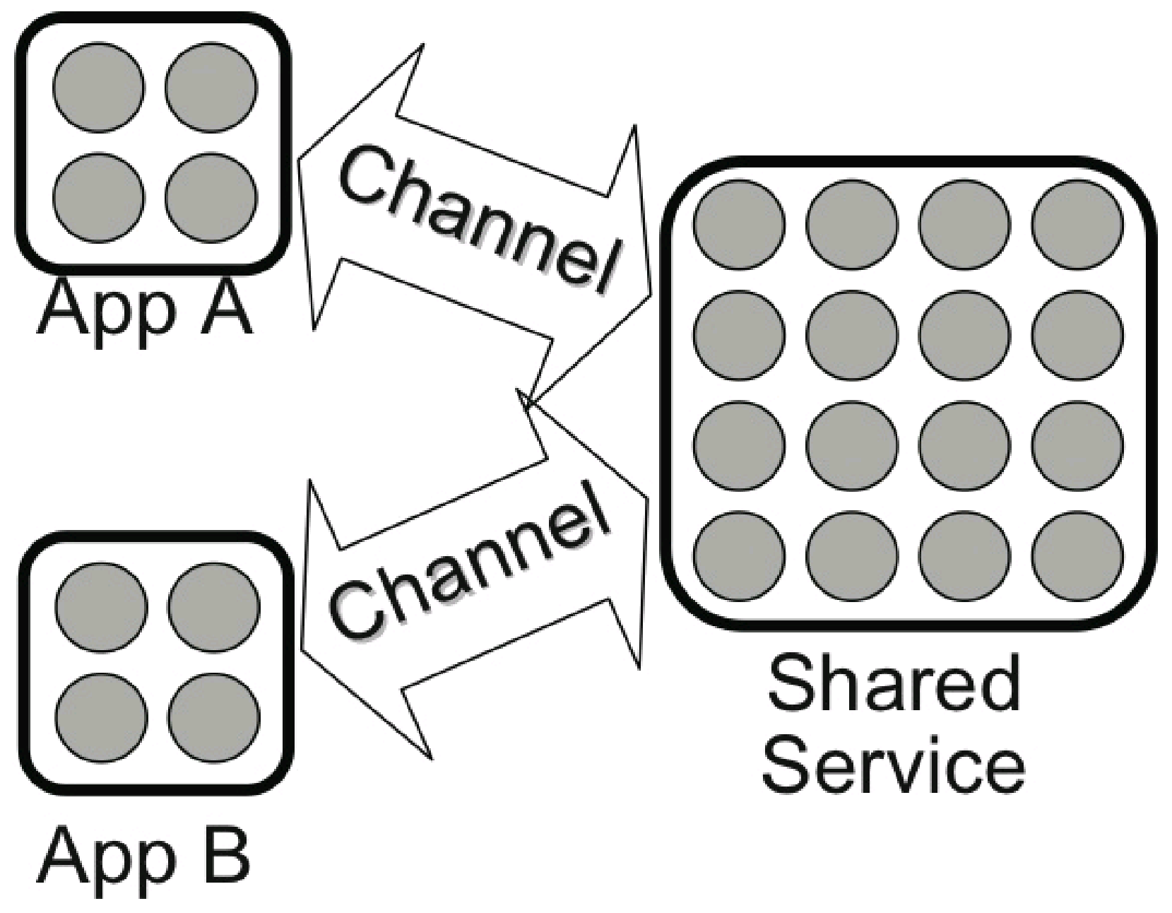
Berkeley ParLab

Porting Qt Embedded To Tessellation OS

Albert Kim, Juan Colmenares, John Kubiatowicz

Improving the Responsiveness of GUI Applications

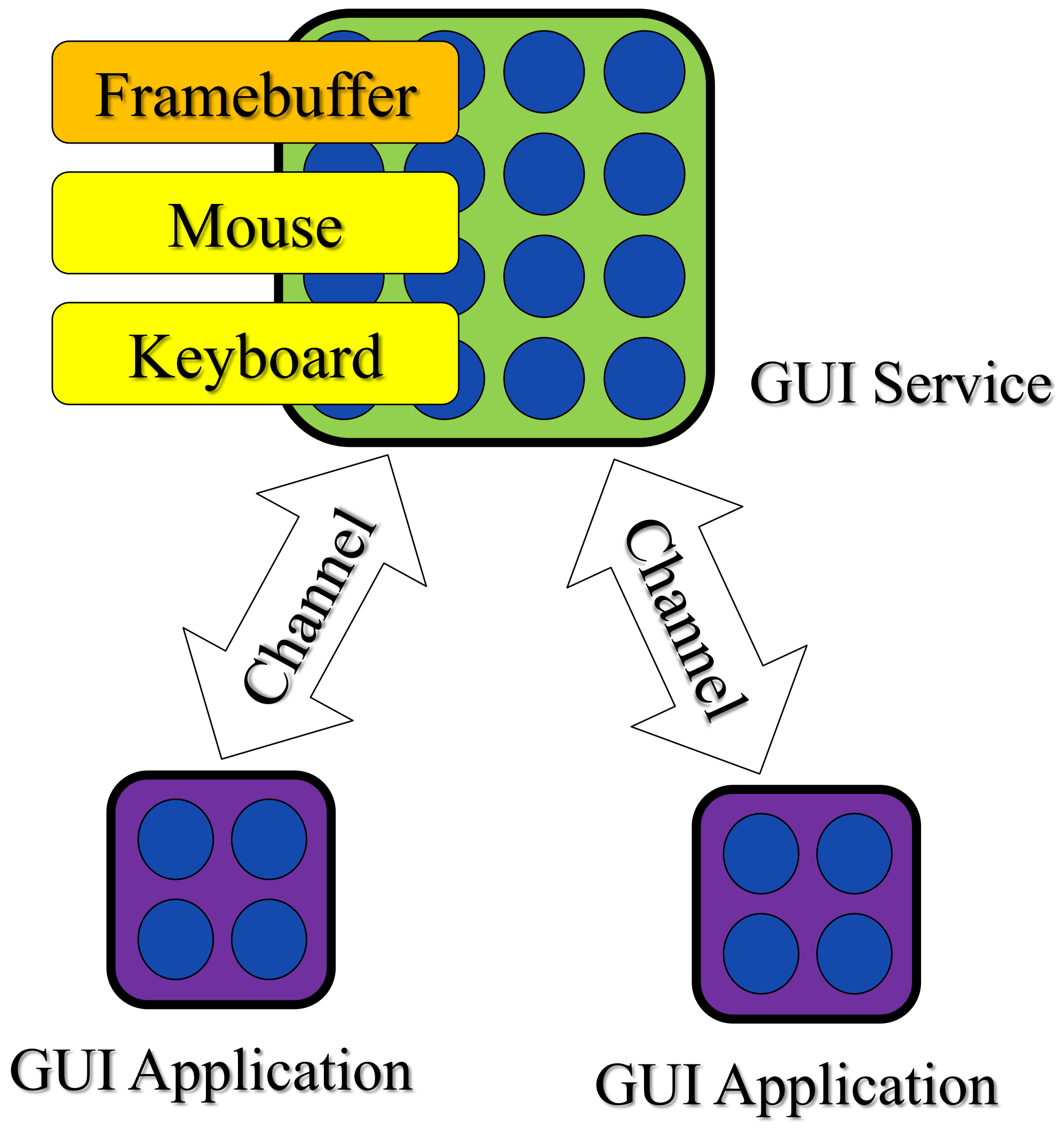
About Tessellation



- Tessellation is a many-core operating system
- Cores are partitioned into cells
  - The cores are partitioned both in space and in time
- Cells may communicate with each other using channels
  - Channels provide asynchronous communication
- Cells that specialize in providing a service are called service cells
  - These often represent interfaces to hardware resources
  - You might have a network service for a network card
- Cells' performances are monitored and the policy service may hand out more resources if performance is too low

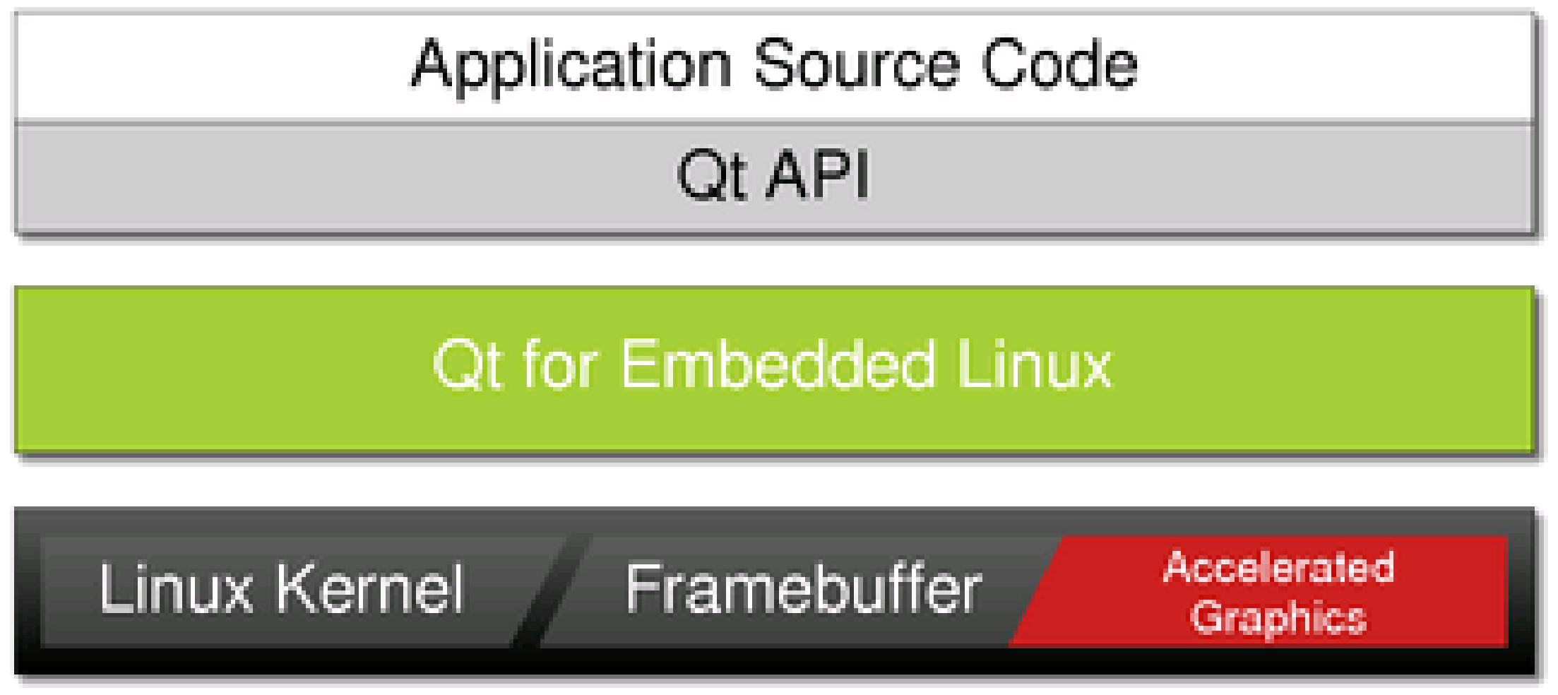
The Plan

- Have one GUI service cell that has unique access to the framebuffer
  - The GUI service will also handle all input sources (e.g. mouse)
- GUI applications will start up in separate cells and contact the GUI service to have their graphics rendered
  - The applications will be guaranteed access to the GUI service at some fixed rate

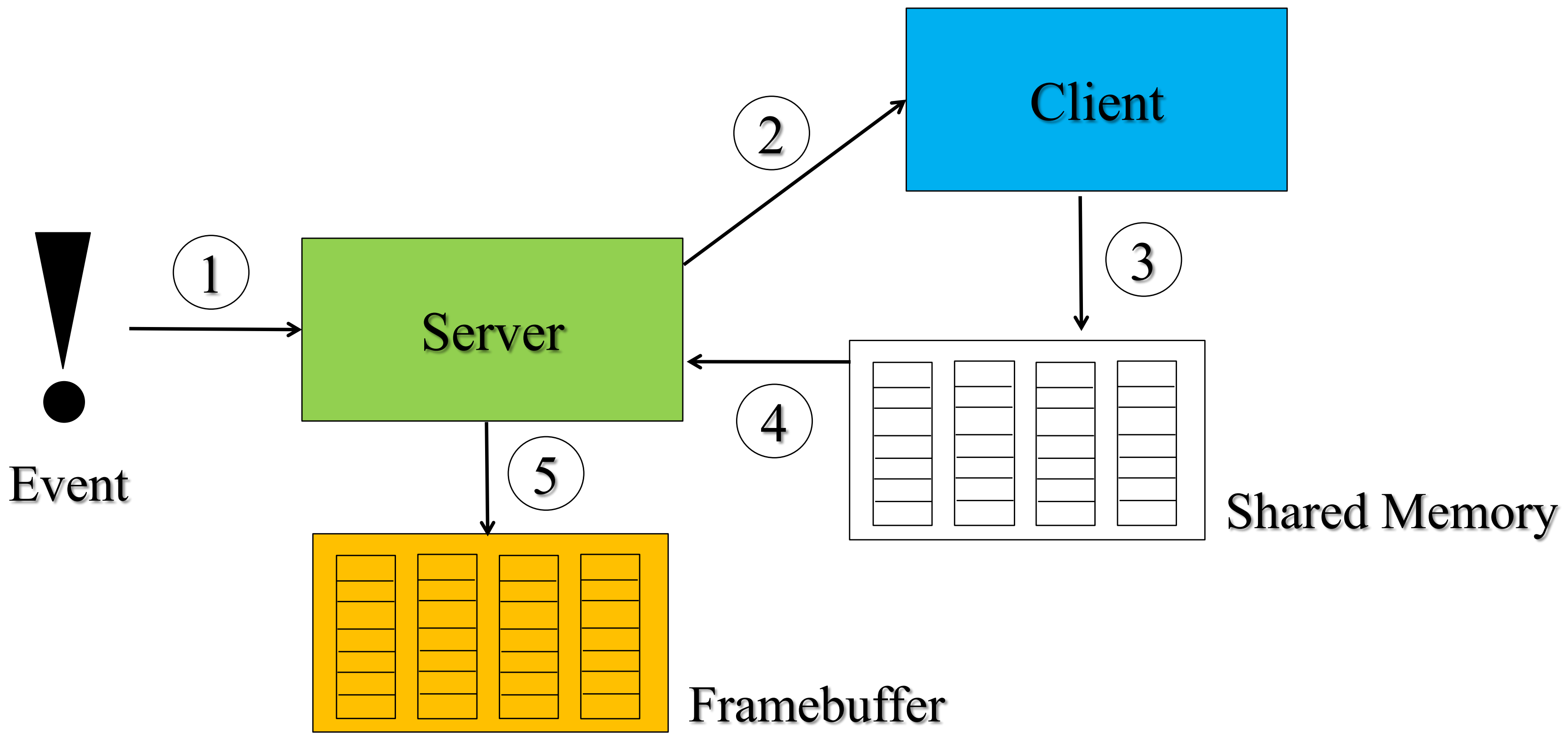


Why Qt Embedded?

- Qt Embedded runs directly on top of the framebuffer (no X!)



- Qt framework is platform-independent
  - Qt applications don't need to be ported; the apps will "just work" on top of ported Qt Embedded
- Qt is very self-contained, so we don't need to port a lot of other libraries
- Qt framework provides many additional libraries that future applications can use
- Qt Embbeded has a client/server model
  1. The server receives an event (e.g. keypress)
  2. The server finds which client it should forward the event to and forwards it using UNIX sockets
  3. The client reacts to the event and usually ends up needing to draw something. The client renders its graphics to the shared memory
  4. The server realizes that a client needs updating and reads from shared memory
  5. The server copies the memory content to the framebuffer device, which renders the graphics on to the screen



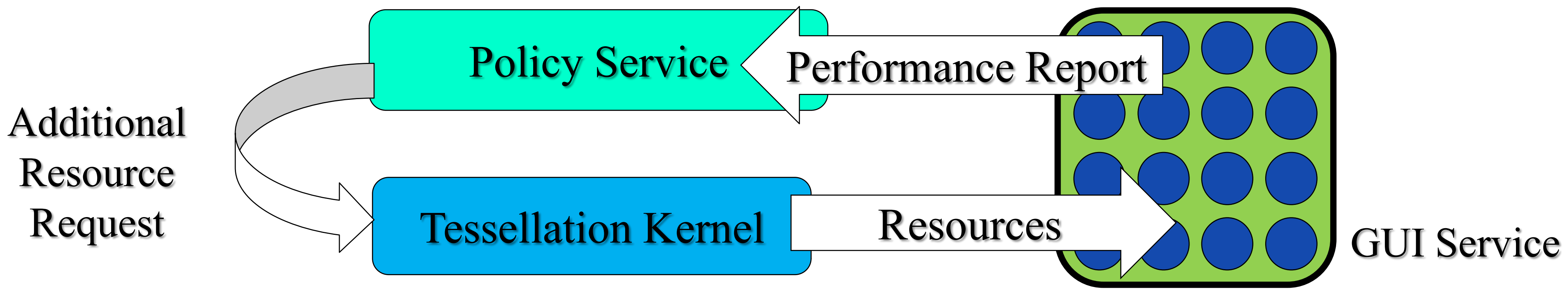
Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227).

Considerations

- Qt Embedded assumes that the underlying OS is very Linux-like (e.g. tty, sockets)
- Qt uses UNIX sockets in the file system to communicate between client and server
  - Tessellation will use channels for this and breaks the UNIX philosophy of doing everything through the file system
- Clients render their graphics on to shared memory and expect servers to read from shared memory
  - Tessellation does not have any plans for shared memory, so channels will be used to transfer the rendered graphics from client to the GUI service
  - Still, shared memory and channels don't have a one-to-one correlation: shared memory is persistent and channels don't need locks to synchronize communication

Interesting Questions

- What resource should the GUI service quantify and guarantee to connecting applications?
  - Paint events take a variable amount of time, so may not be good
  - CPU time is a possible option
  - Number of pixels drawn is another option
- For a GUI service that uses multiple cores, how should it schedule the cores so that the work can be parallelized correctly?
  - Assigning a core per application is a simple solution, but is not scalable
  - Perhaps each core can handle a part of the screen
- The clients do the actual rendering, so how can we parallelize the client?
  - Qt framework already divides the total area that is to be painted into multiple paint jobs; we can split these jobs on to separate cores, but then we need to write to the channel in a synchronous manner
- How can GUI cells adapt to limited resources and report it adaptation to the policy service in a way that makes sense?
  - Shadowing on images may be considered a luxury and may only be turned on if there are enough cores available


Policy Service
Performance Report
Additional
Resource
Request
Tessellation Kernel
Resources
GUI Service