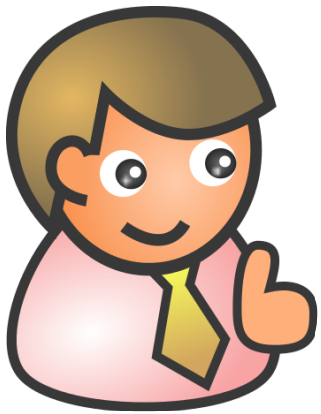
Multi-cloud and cloud-desktop coordination made simple by GXP on Azure

Kenjiro Taura and Ting Chen

University of Tokyo

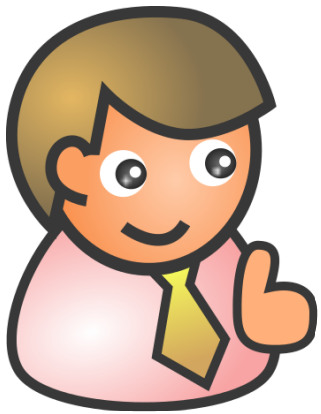
Background

- Clouds becoming platform of choice for workflows, but ...



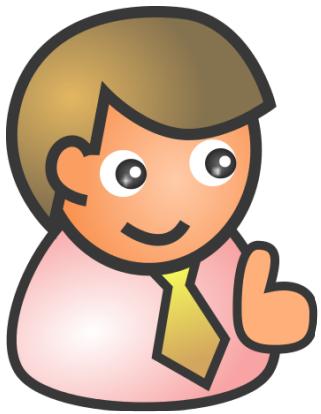
Background

- Clouds becoming platform of choice for workflows, but ...
  - Each cloud platform has its idiosyncrasies



Background

- Clouds becoming platform of choice for workflows, but ...
  - Each cloud platform has its idiosyncrasies
  - Real apps are complex (Java, Perl, Ruby, Python, ...)

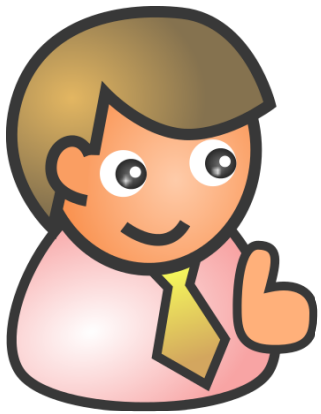


Background

- Clouds becoming platform of choice for workflows, but ...
  - Each cloud platform has its idiosyncrasies
  - Real apps are complex (Java, Perl, Ruby, Python, ...)
  - Clouds are only part of the picture; users have desktops, laboratory clusters, and center machines (supercomputers)

Background

- Clouds becoming platform of choice for workflows, but ...
  - Each cloud platform has its idiosyncrasies
  - Real apps are complex (Java, Perl, Ruby, Python, ...)
  - Clouds are only part of the picture; users have desktops, laboratory clusters, and center machines (supercomputers)
- GXP is a rescue!

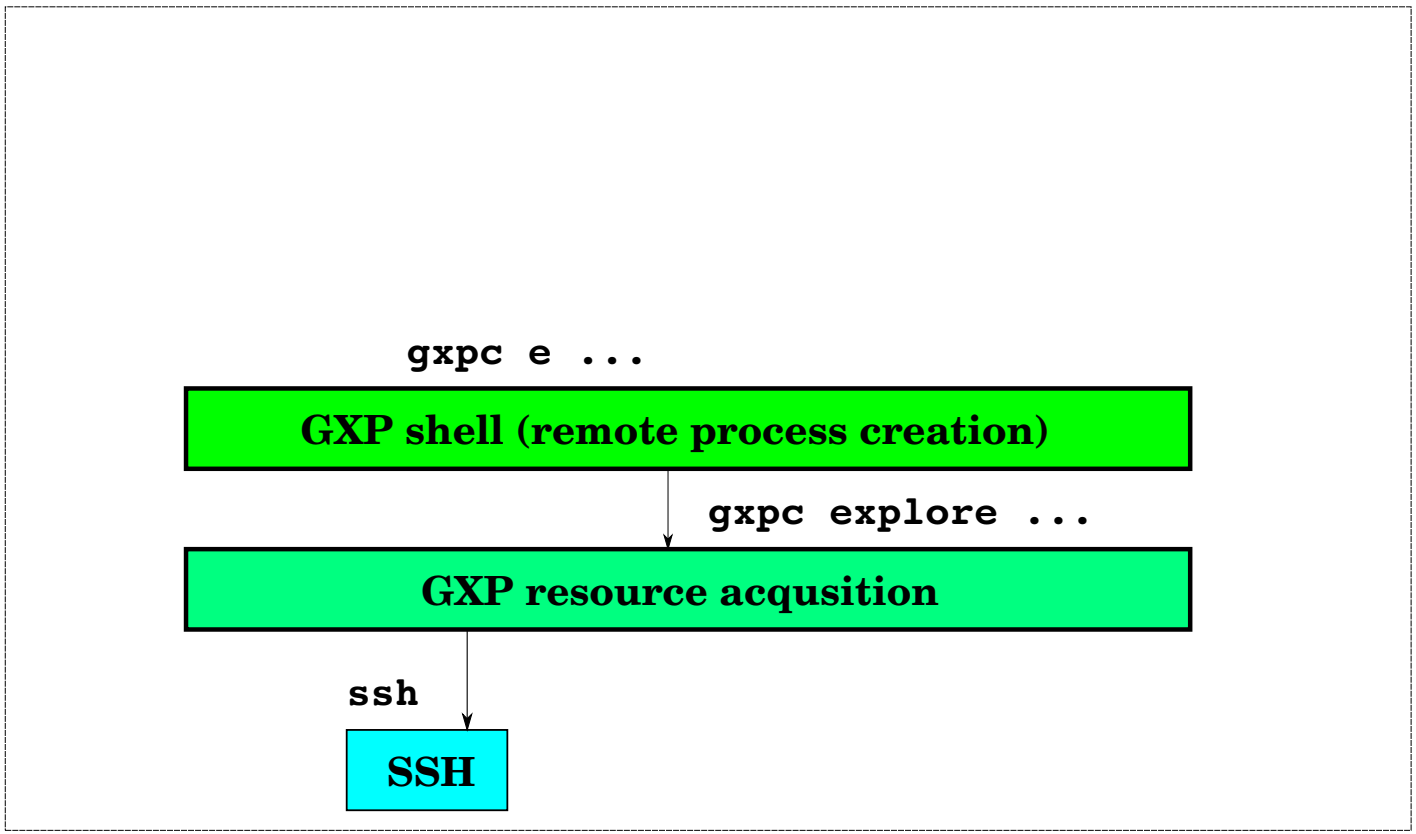


This talk

- GXP : what is it?
- Adapting GXP to Azure
- Wrap-up

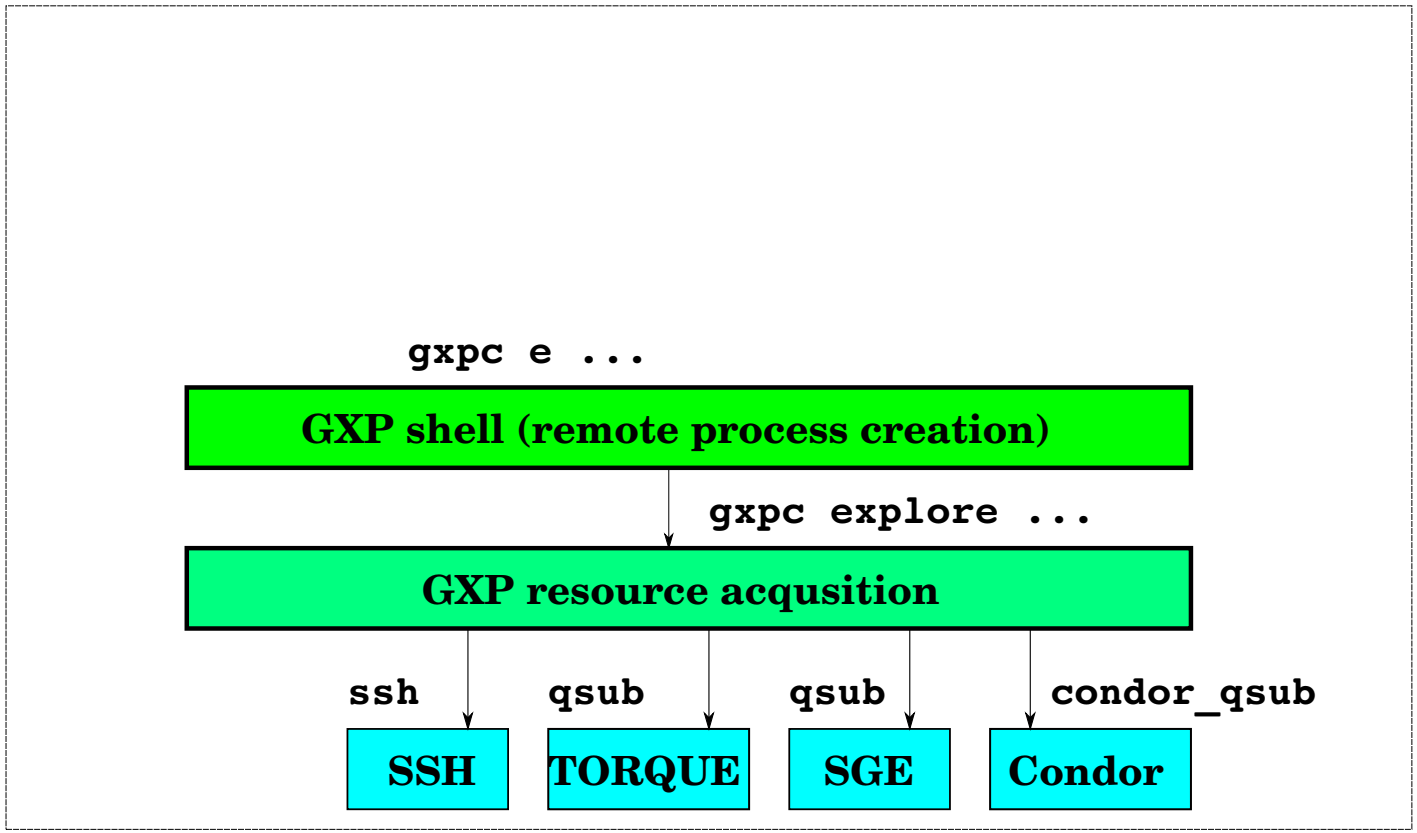
GXP small history

(2005∼) Initially developed as a parallel shell across multiple clusters on top of SSH



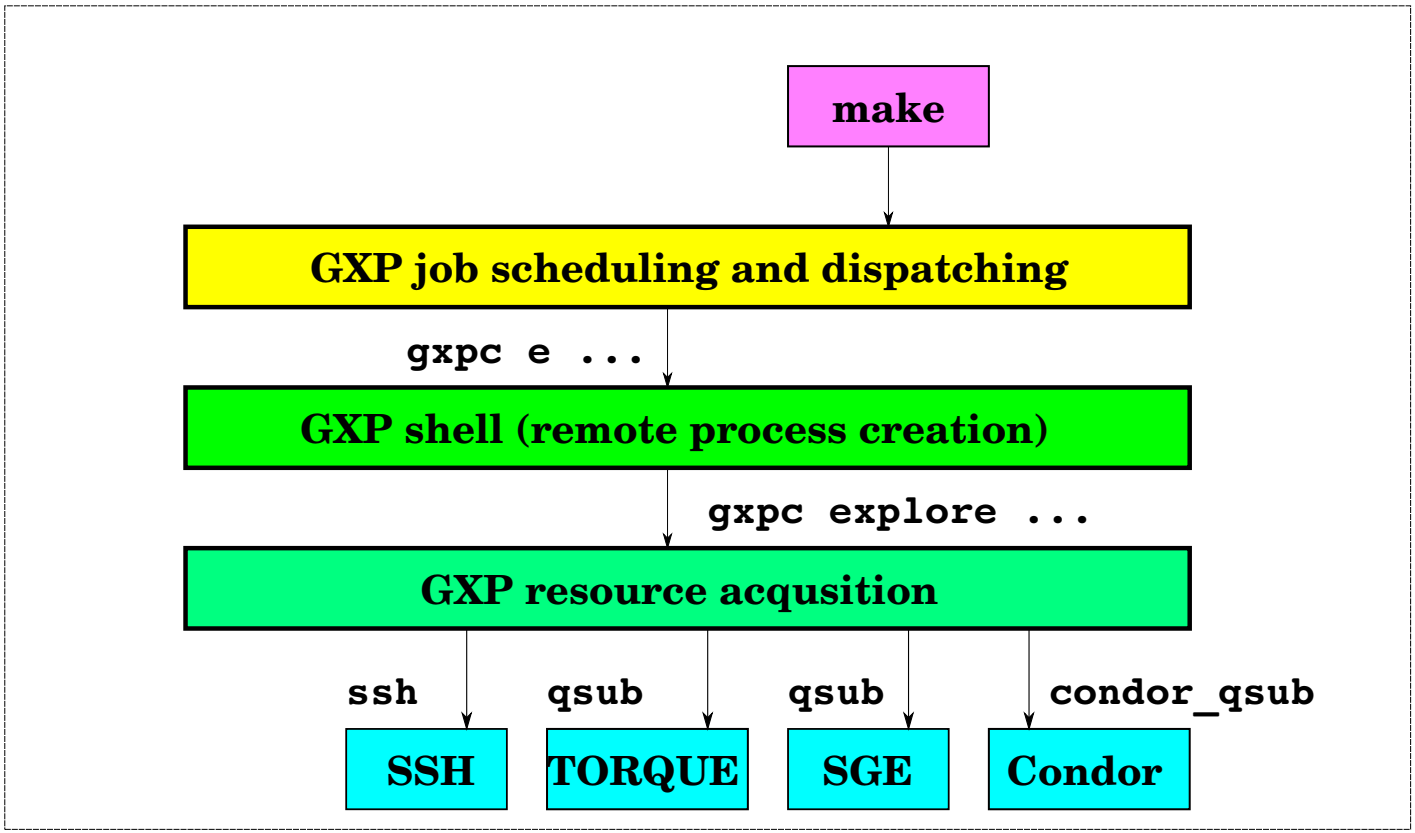
GXP small history

(2006∼) The identical interface on top of batch schedulers



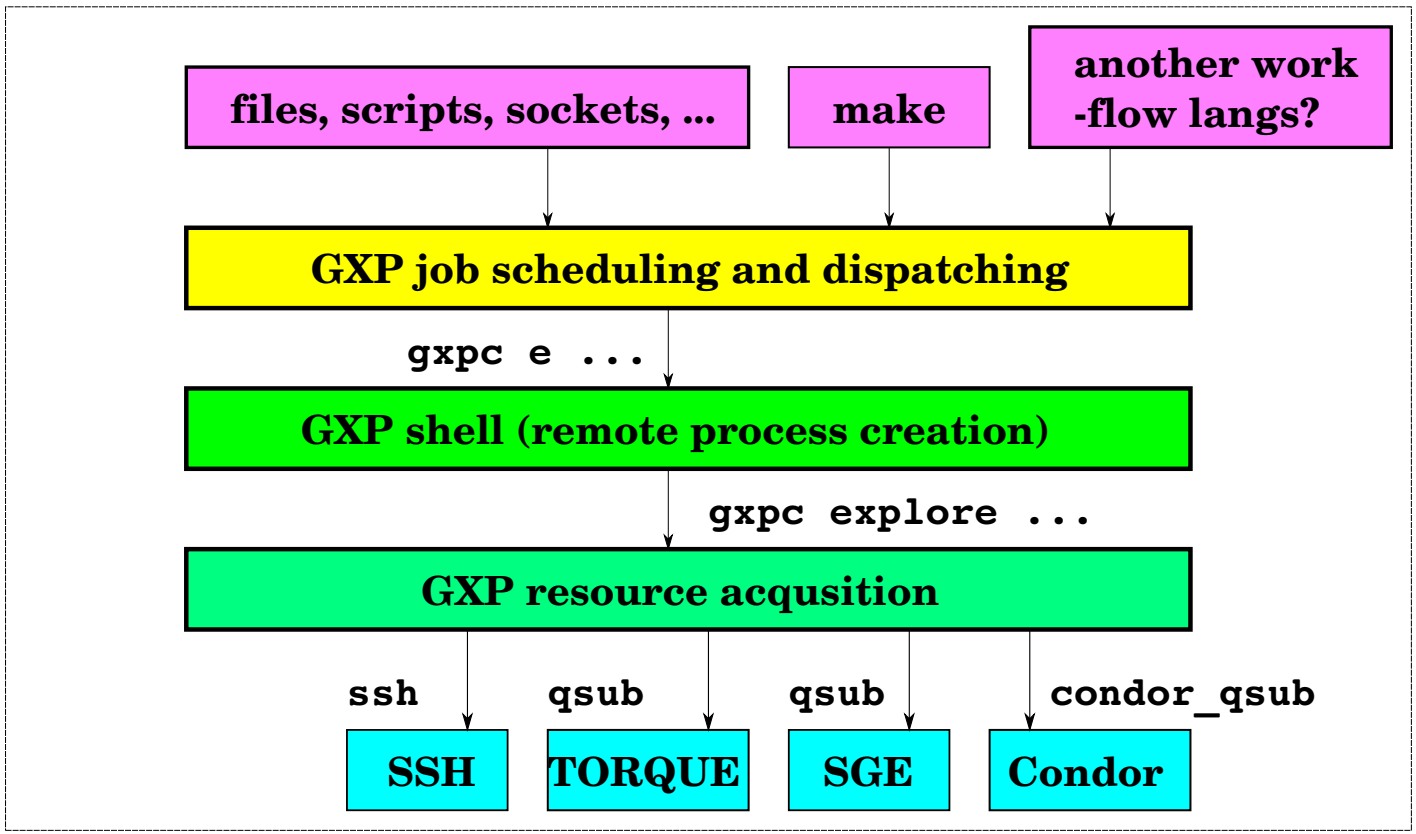
GXP small history

(2008∼) Workflows based on Makefiles



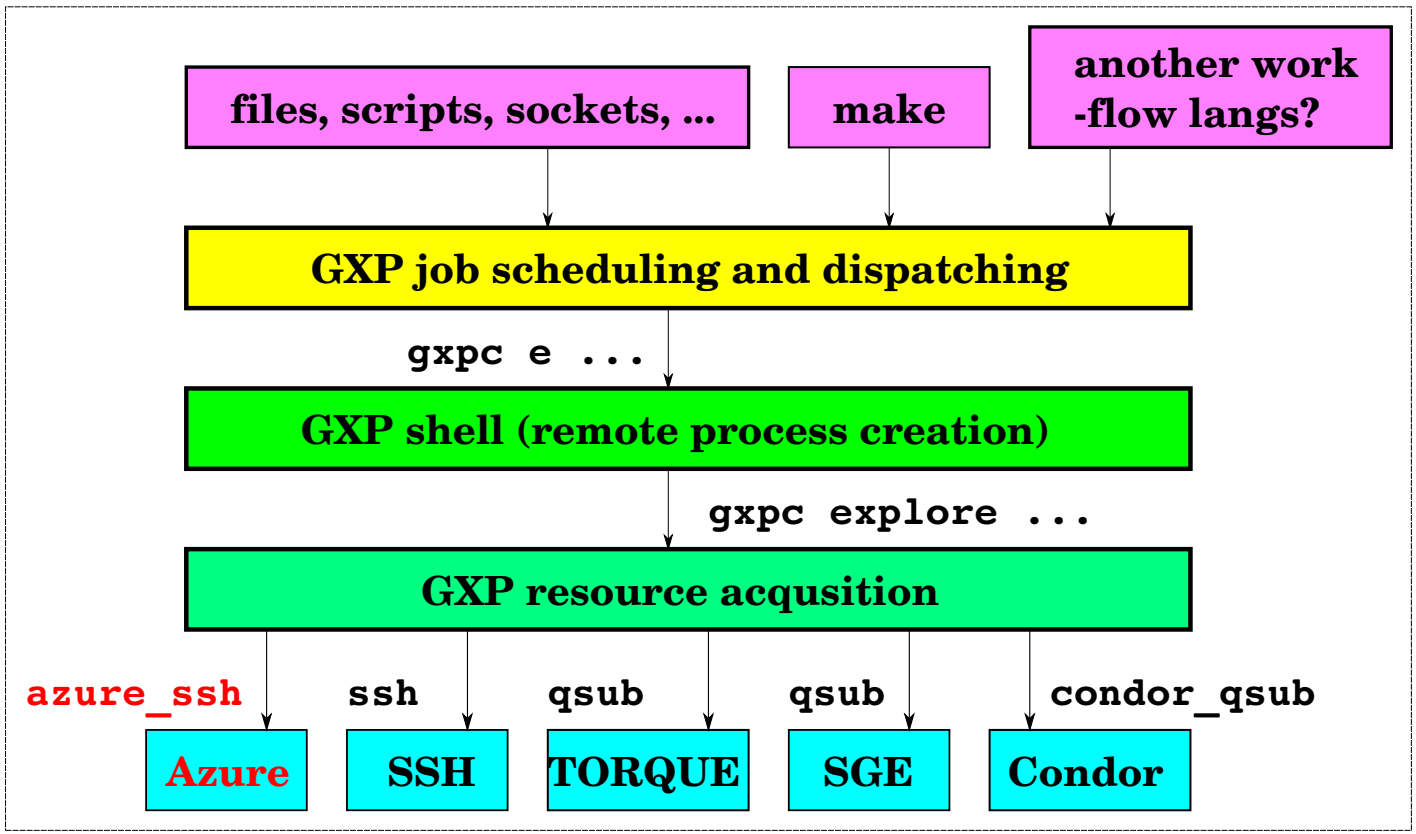
GXP small history

(2010∼) Job scheduling and dispatching frameworks to support arbitrary frontend



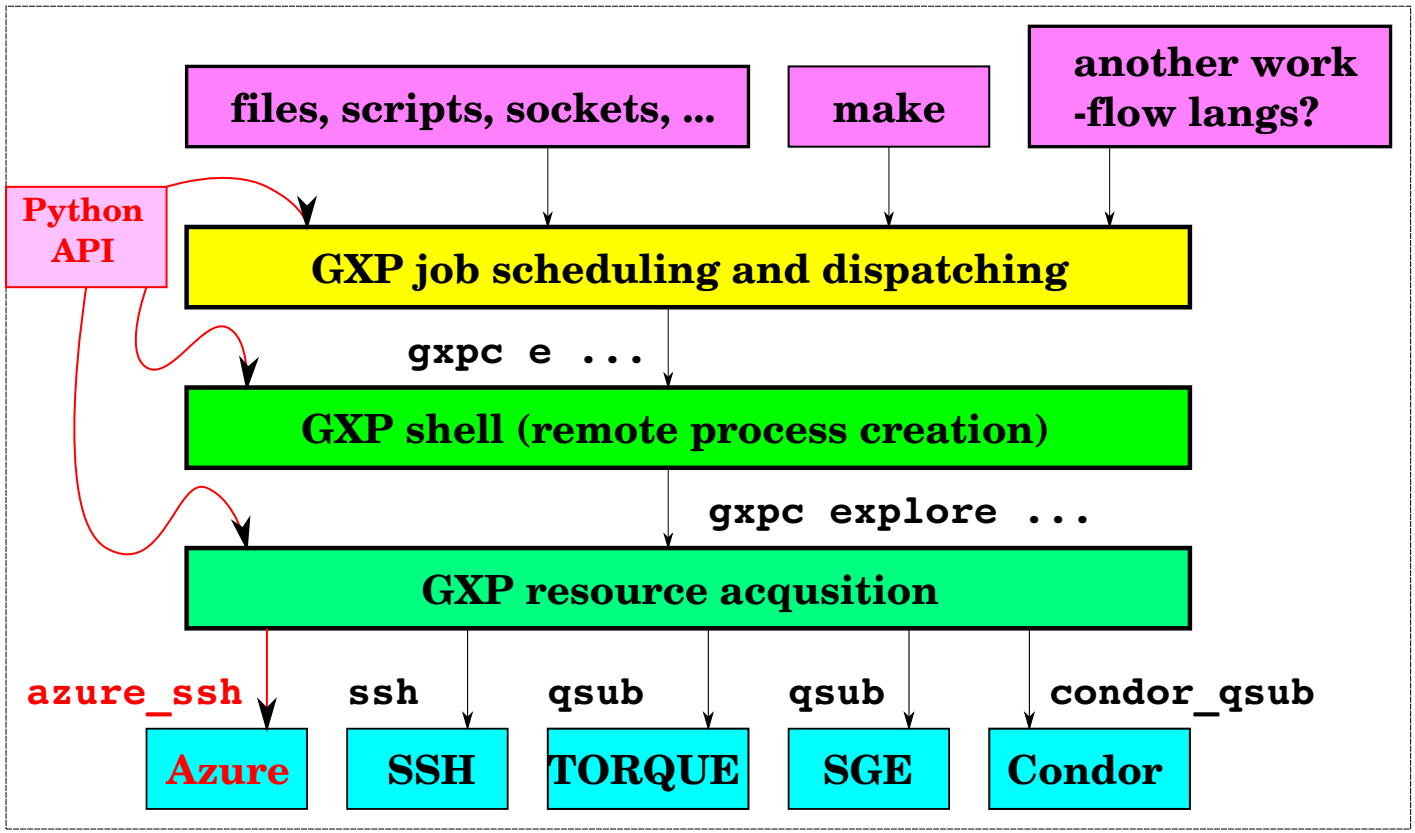
GXP small history

(2010∼) Job scheduling and dispatching frameworks to support arbitrary frontend



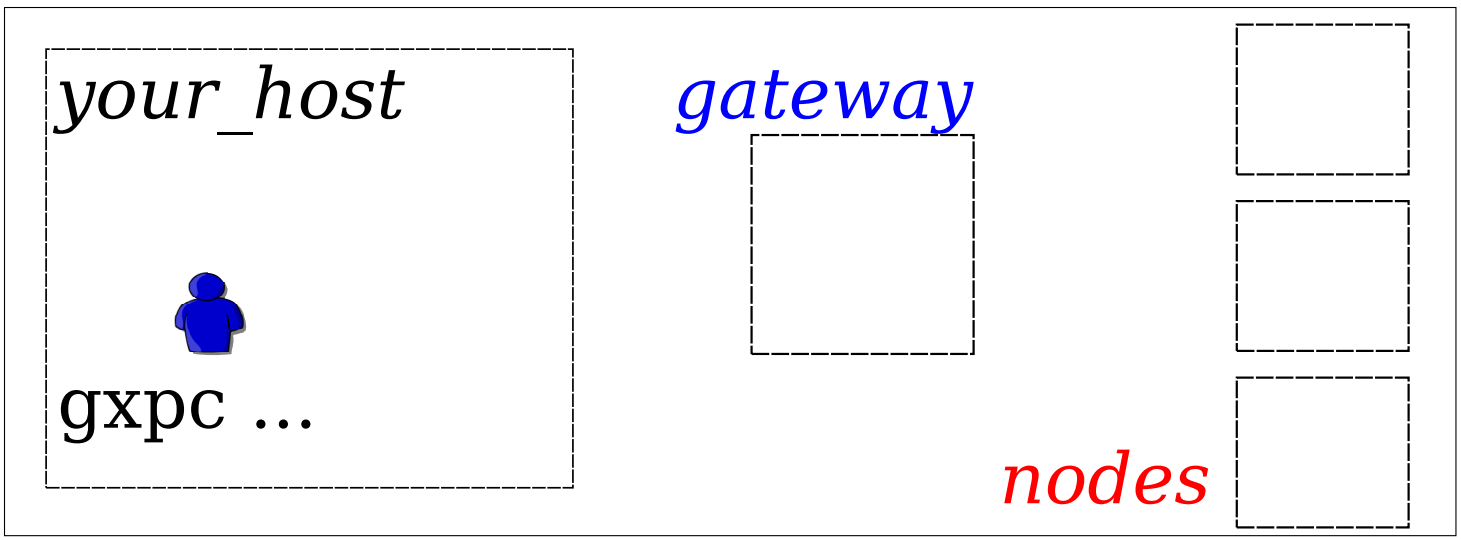
GXP small history

(2010∼) Job scheduling and dispatching frameworks to support arbitrary frontend



GXP user interface

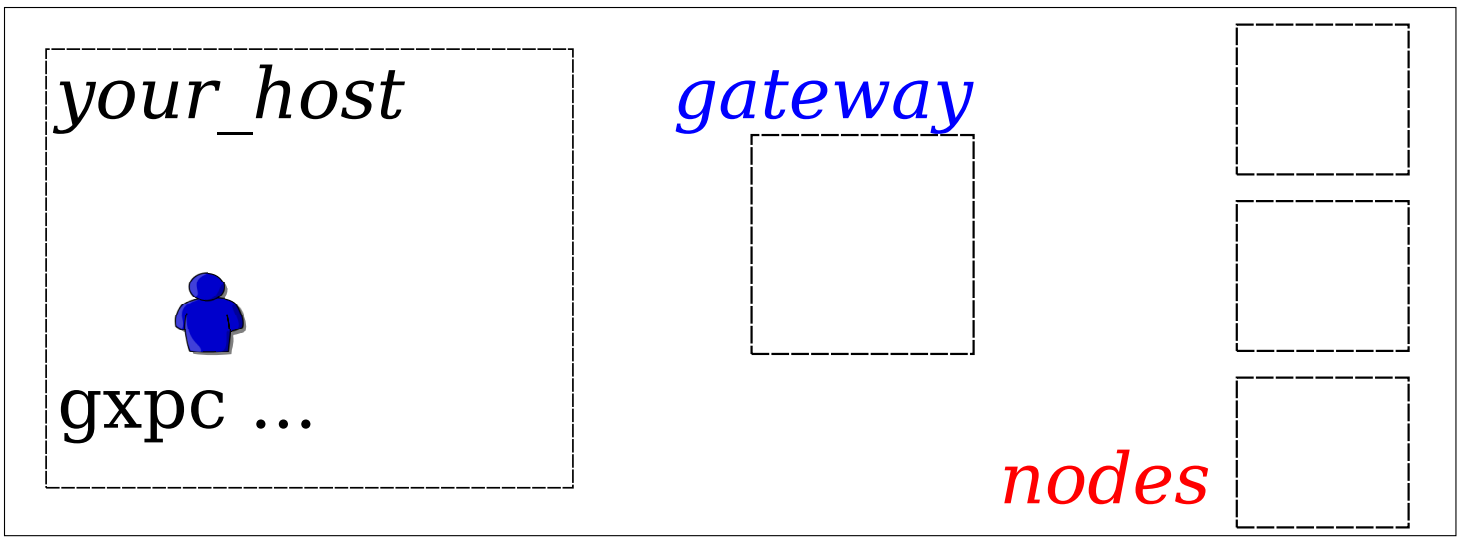
- `use, explore`, and execute



GXP user interface

- `use, explore`, and execute

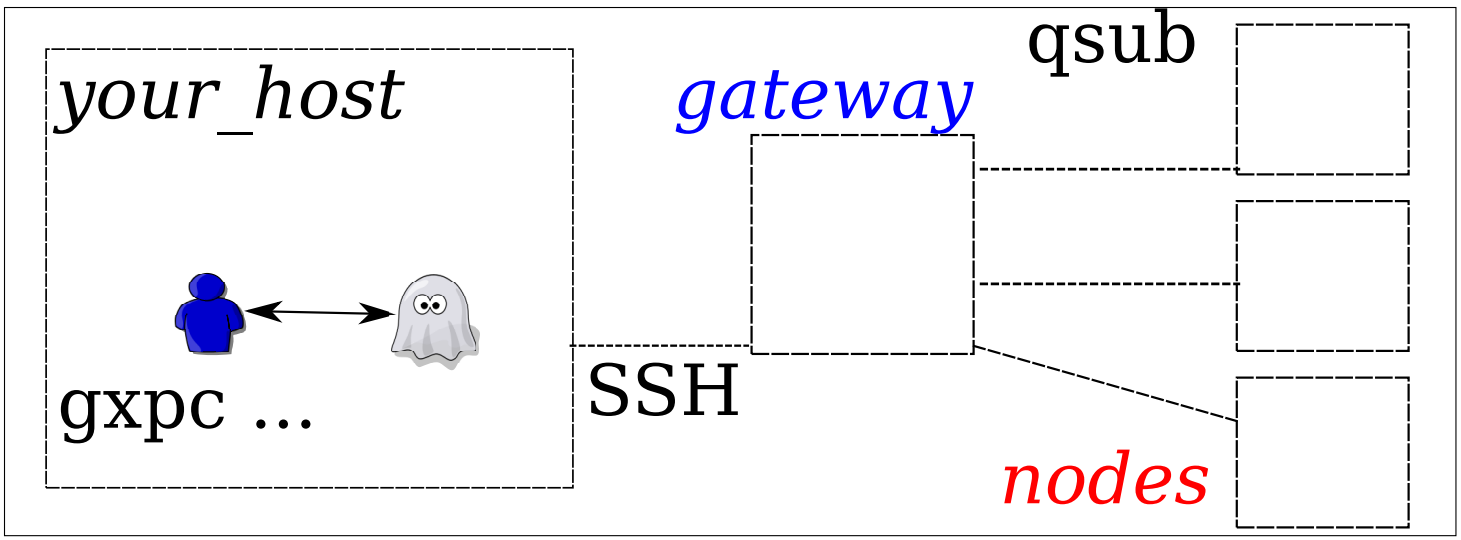
```
gxpc use ssh     your_host gateway
gxpc use torque  gateway   node
```



GXP user interface

- `use`, `explore`, and execute

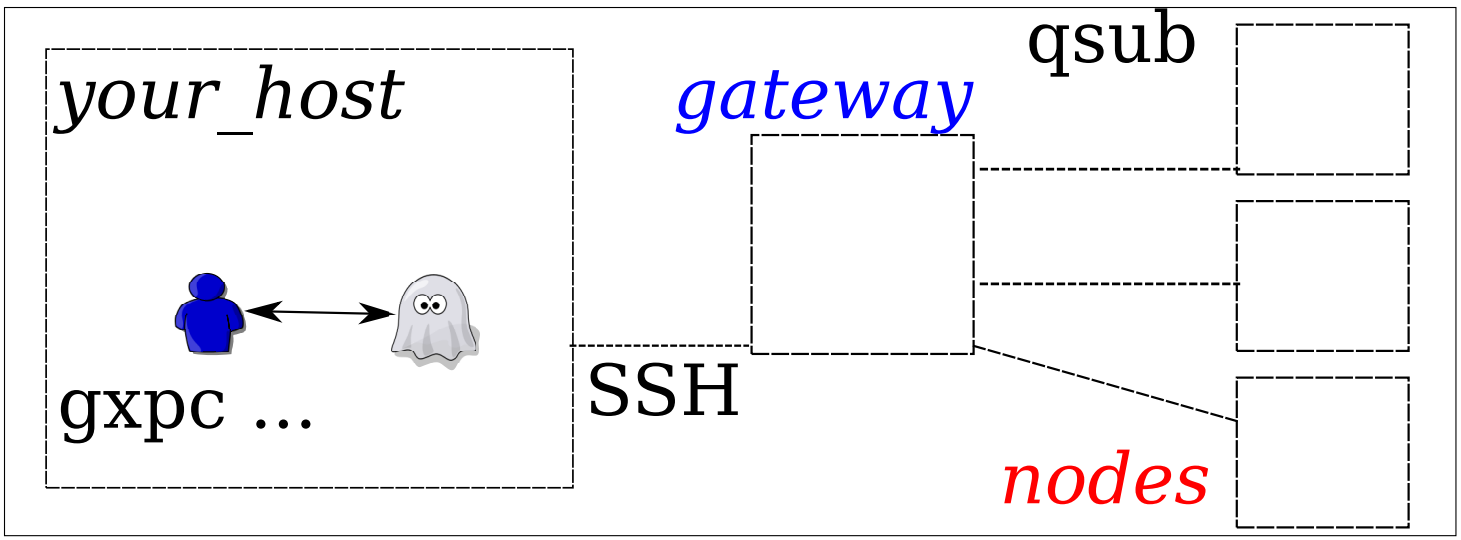
```
gxpc use ssh     your_host gateway
gxpc use torque  gateway   node
```



GXP user interface

- `use`, `explore`, and execute

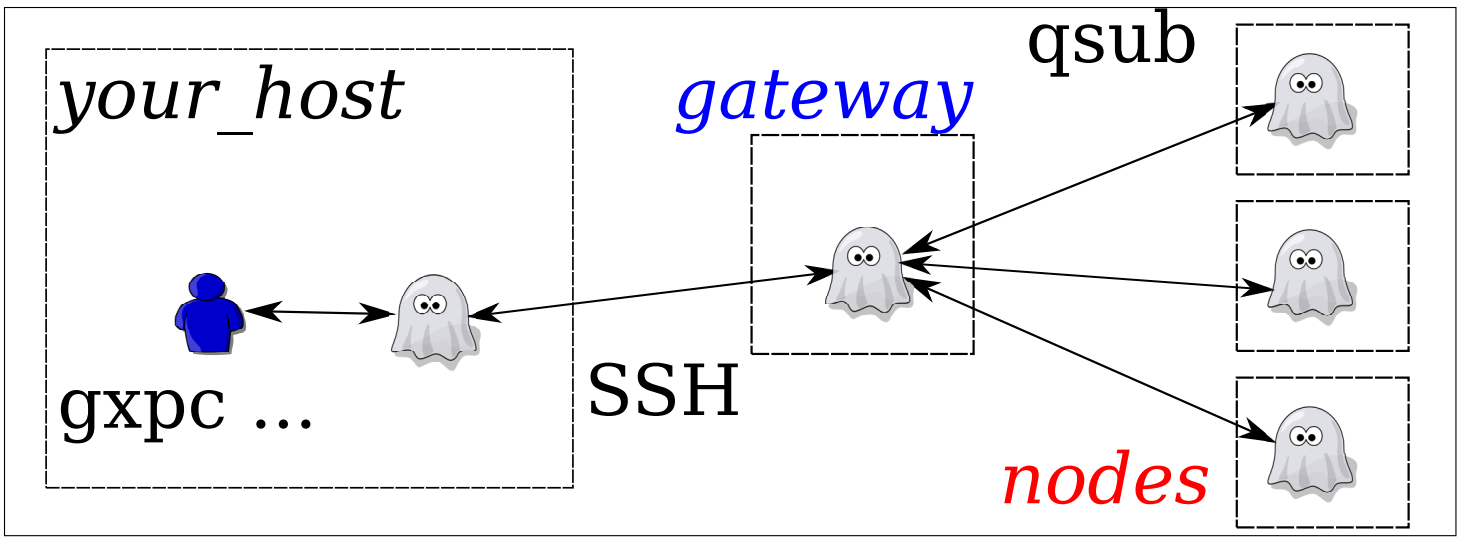
```
gxpc use ssh      your_host gateway
gxpc use torque   gateway   node
gxpc explore      gateway   node[[000-099]]
```



GXP user interface

- `use`, `explore`, and execute

```
gxpc use ssh      your_host gateway
gxpc use torque   gateway   node
gxpc explore      gateway   node[[000-099]]
```



GXP user interface: highlights

- Flexible: It accommodates many ways to connect to remote resources and restrictions

GXP user interface: highlights

- Flexible: It accommodates many ways to connect to remote resources and restrictions
- Quick and uniform: It is only when you 'explore' that it issues remote shell or job submission commands

GXP user interface: highlights

- Flexible: It accommodates many ways to connect to remote resources and restrictions
- Quick and uniform: It is only when you 'explore' that it issues remote shell or job submission commands
- No burden to install: The 'explore' automatically *bootstraps* remote GXP daemons, without assuming its prior installation

Collaboration and use cases

- U-Tokyo Tsujii group
  - Indexing the whole PubMed abstracts for Medie semantic search engine (9,000 cores)
  - Event extraction from the whole PubMed abstracts (9,000 cores)
- U of Manchester (NaCTeM)
  - Indexing PubMed full papers for semantic search engine (8,192 cores)
- U-Tokyo Morishita group
  - Human genome read alignments for UTGB genome browser (8,192 cores)
- Kyoto U, NICT

This talk

- GXP : what is it?
- Adapting GXP to Azure
- Wrap-up

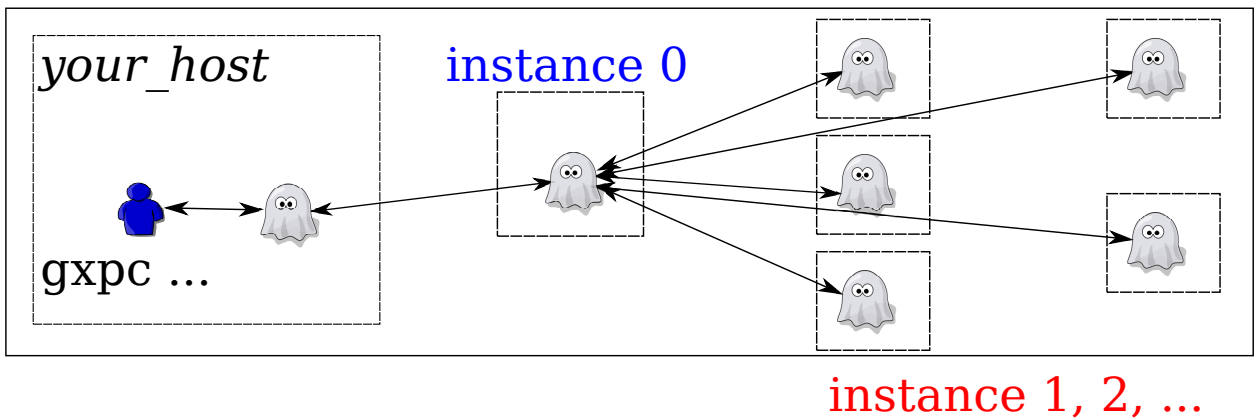
Adapting GXP to Windows Azure

1. User interface design
2. Bootstraping GXP daemons on Azure
3. Porting GXP core functions to Windows

GXP on Azure: user interface

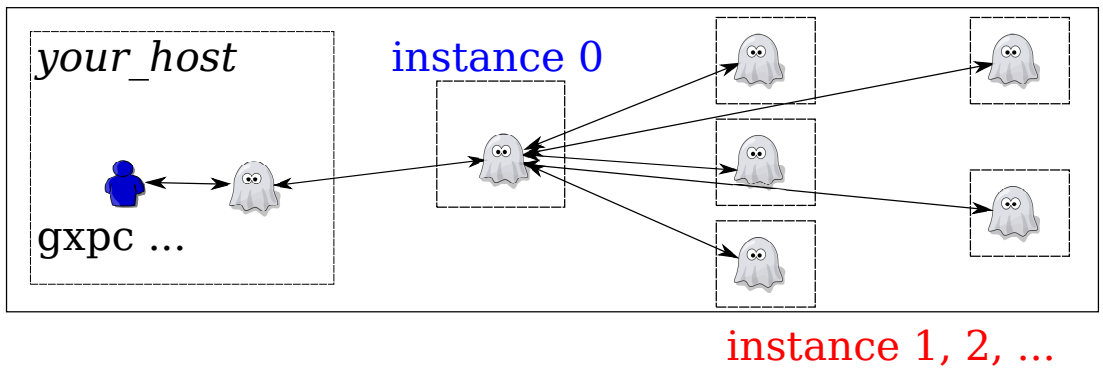
- **Step 1:** Launch instances of a worker role
- **Step 2:** With that, Azure is just another resource type with a particular naming convention

```
gxpc use azure    your_host           service_name:port
gxpc use azure_i  service_name:port   azure_instance
gxpc explore      service_name:port   azure_instance[[1-5]]
```



Bootstrapping GXP daemons on Azure

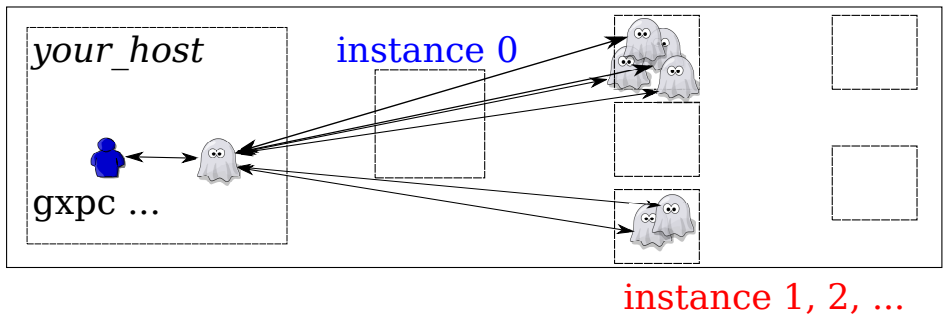
- We need an equivalent of *"rsh daemon"* on Azure
- The rest of the bootstrapping process already in GXP
- $\Rightarrow$ Spin up N worker role instances each acting like an rsh daemon, *and bootstrap a single GXP daemon on each instance*



Turned out it's not trivial, due to firewalls and the Azure load balancer

Azure issues (1)

- *Socket communication on arbitrary TCP/UDP ports not allowed, even internally*
  - $\Rightarrow$ *list intended ports as endpoints* in the service definition
- Azure load balancer redirects connection from outside Azure to any instance. *We don't know which one will serve a connection to it.*



There are two logical workarounds...

Azure issues (2)

- **Workaround 1:** *Maintain only one worker listens on an endpoint*
  - ⇒ Only one instance (ID= 0) listens on an input endpoint
  - There is a limit on the number of endpoints (25 per service). So, the luxury of allocating a distinct input endpoint for each instance not allowed
- **Workaround 2:** *Somehow specify which instance you connect to*
  - Works once you are inside Azure (map instance ID to its *ip_addr:port* via Service Runtime API)
  - ⇒ All instances except ID= 0 listen on an internal endpoint. Each is identified by its *ip_addr:port*

Service def/config (1)

- We use a worker role with native code execution

```
<ServiceConfiguration serviceName="azure_rsh" ...>
  <WorkerRole name="AzureRshd"
    enableNativeCodeExecution="true">
```

- The number of instances is arbitrary

```
<Instances count="10" />
```

Service def/config (2)

- Allocate an input port and an internal port

```
<InputEndpoint name="ExternalEndpoint"
   protocol="tcp" port="10000" />
<InternalEndpoint name="InternalEndpoint"
   protocol="tcp" />
```

The entry point (C# code)

```
Run() {
    get all ip_addr:port of the internal endpoint;
    put them in an environment variable;
    run "python.exe azure_rsh.py"; // do real work here
    wait it to finish;
}
```

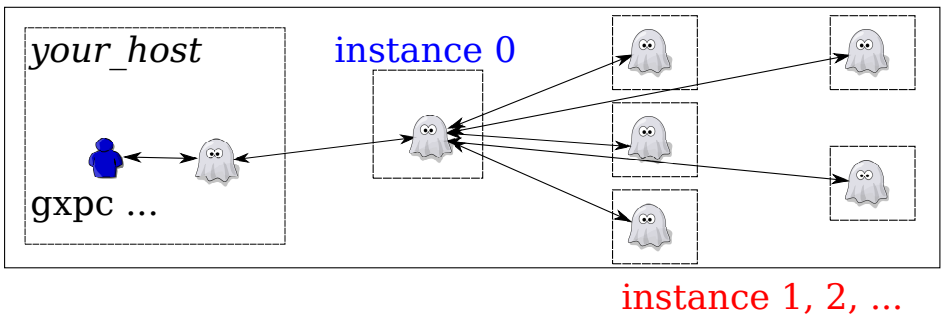
Azure RSH (Python code)

```
def main():
    while true:
        if instance_id == 0:
            port = port of the input endpoint
        else:
            port = port of the internal endpoint
        so = accept_conection(port)
        cmd = get_command_line(so)
        spawn_process_with_pipes(cmd)
        handle_stdin_stdout_stderr()
```

- Just a trimmed-down version of rsh server
- *Is there an SSH server or alike available out of the box?*

Porting GXP core functions to Windows

- It is written in the single thread event-driven style (select loop à la Unix)



- GXP daemon is an event-driven application that handles
  - communication with gxpc frontend (sockets)
  - communication with other daemons (pipes)
  - communication with local subprocesses (pipes)
  - death of subprocesses
  - signals

Porting event-driven loops is always tricky ...

Porting GXP core functions to Windows

- Windows has select, but only for sockets (not for pipes)
- Port on cygwin was unreliable
- Python Win32 extension was the rescue
- `WaitForMultipleObjects` is the native API to watch for multiple events
  - pipes → named pipes and overlapped I/O
  - sockets → WSAEventSelect
- *Is there a workaround for* `WaitForMultipleObjects` *taking only up to 64 handles?*

This talk

- GXP : what is it?
- Adapting GXP to Azure
- Wrap-up

Current implementation status

- GXP daemon brings up locally on Windows
- Porting frontend and the job dispatcher on the way (should be much easier than daemons)
- Initial testing of Azure RSH done
- Testing Windows Azure Drive on the way

Plans ahead

- Collaborating with Tsujii's group
  - Run the PubMed indexing workflows on Azure
  - Make GXP on Azure with their NLP tools available to the community
- Data sharing/exchange between jobs
  - Shared Azure Drive as an initial attempt
  - Workflow system stage them between blob and local Azure Drive
  - *Parallel file system for Windows?*