Thanks, Costin !

# Xcrypt
# Highly-Productive Parallel Script Language

Hiroshi Nakashima
(ACCMS, Kyoto U.)

with the cooperation o
Takeshi Iwashita and
Tasuku Hiraishi

who is really working

T2K Open Supercomputer Alliance

## Contents

- Yet Another HPC Programming
  - Not Only by XcalableMP
  - Capability and Capacity Computing with Script Language
  - PDCA (Plan-Do-Check-Action) Cycle
- HP&P Script Language
  - Design Goal and Concept
  - Language System Architecture
  - Current Status and Near Future Work
- Concluding Remark

T2K Open Supercomputer Alliance

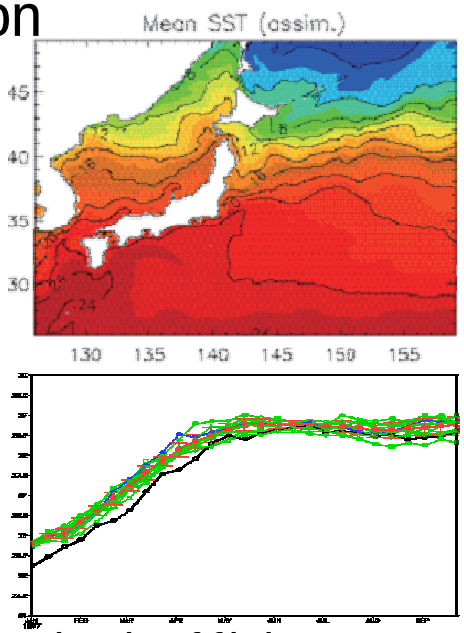
## Yet Another HPC Programming
# Not Only by XcalableMP

- Use of a HPC system for R&D ...
  - is not just a single ru[plan-do-check-action]ram
  - but has many PDCA cycles with many runs
- HPC application programming ...
  - is not limited to from-scratch with Fortran, C(++), Java, ... and with MPI, OpenMP, XMP...
  - but includes glue-programming for;
    - do-parallel executions of a program
    - interfacing programs and tools
    - PDCA cycle management
    - ...

T2K Open Supercomputer Alliance

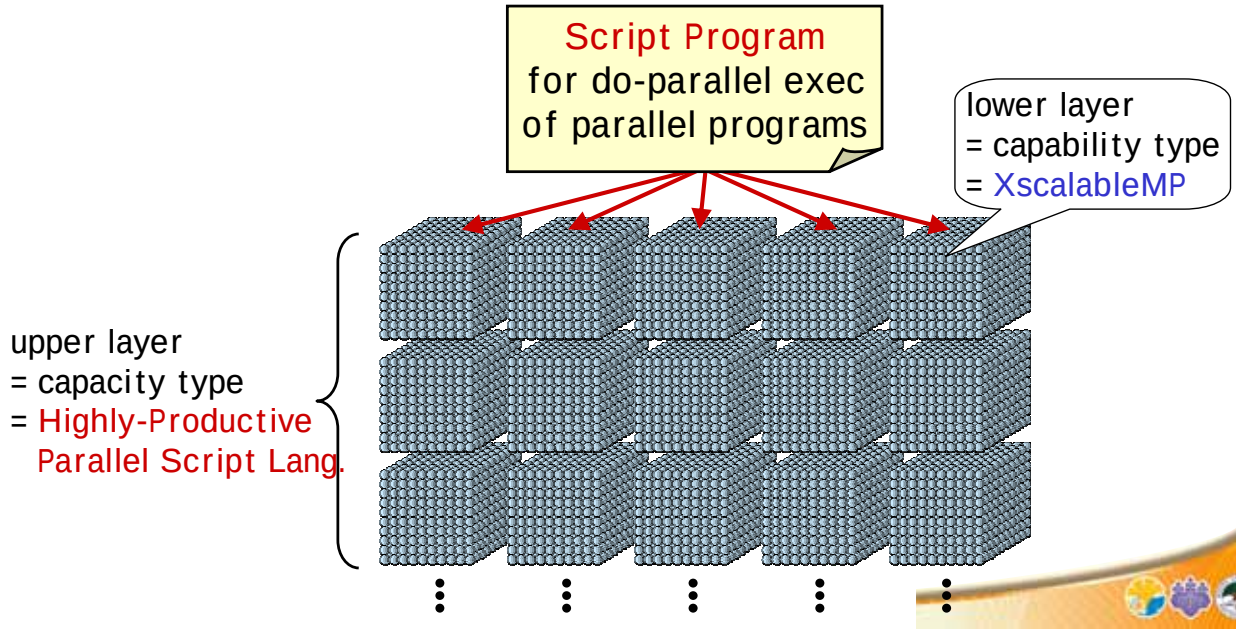
## Yet Another HPC Programming
# Example of C&C Computing

- Oceanographic Simulation
  - Capability Computing
    - Navier-Stokes + Convective Heat Xfer + ....
    - Fortran + MPI, of course
  - Capacity Computing
    - Ensemble Simulation with various initial/boundary conditions
    - Fortran + MPI, why???
      ➔Not only unnecessary but also inefficient
    - Do it with Script Language !!!



## Yet Another HPC Programming
# C&C with Script Language

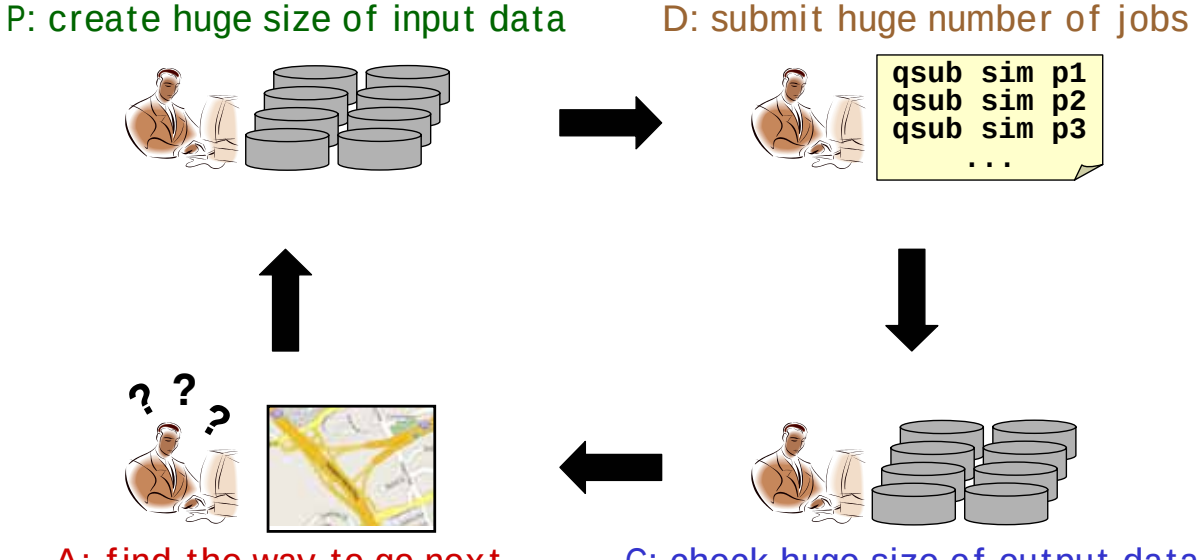
- Two-Layered Million-Scale Programming
  ← $10^3$ capability x $10^3$ capacity = $10^6$





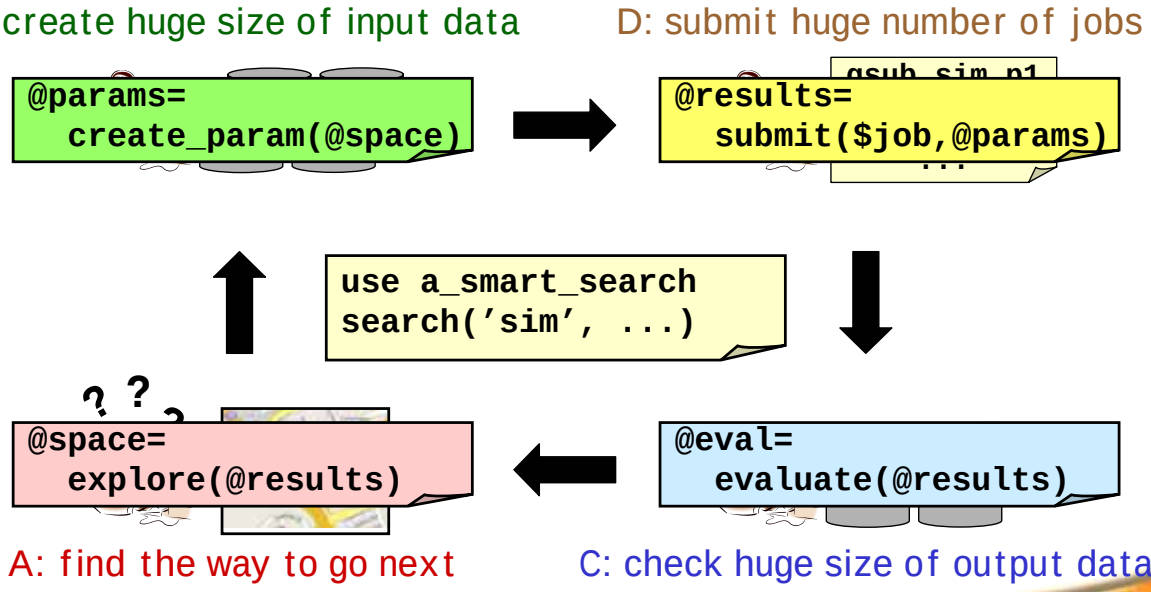

## Yet Another HPC Programming
# Goal=Automated PDCA Cycle

- e.g. Ensemble-Based Data Assimilation
  = repeated sim to find opt parameter



## Yet Another HPC Programming
# Goal=Automated PDCA Cycle

- e.g. Ensemble-Based Data Assimilation
  = repeated sim to find opt parameter






## HP&P Script Language
# Why HP & P

- Script Language
  - inherently suitable for programming to run programs
  - rich functionality for gluing programs
  - easy-to-write for computer scientists
- Parallel Script Language
  - functions to run programs in parallel
    e.g. submit many jobs and wait for their completion
- Highly-Productive Parallel Script Language
  - easy-to-write for computational scientists
    - create input files from a template easily
    - extract desired lines/words from output files easily

HP&P Script Language

# Design Goal

- Easy-to-Write
  - even for guys who never hear of regular expressions, object-oriented, ...
  - not requiring more than 10 lines for simple parameter sweeps.
- Rich Functionality
  - to implement easy-to-write magic by wizards who supports Muggles.
  - to glue applications and GUI (if you love it), visualization tools, data capture tools, ...



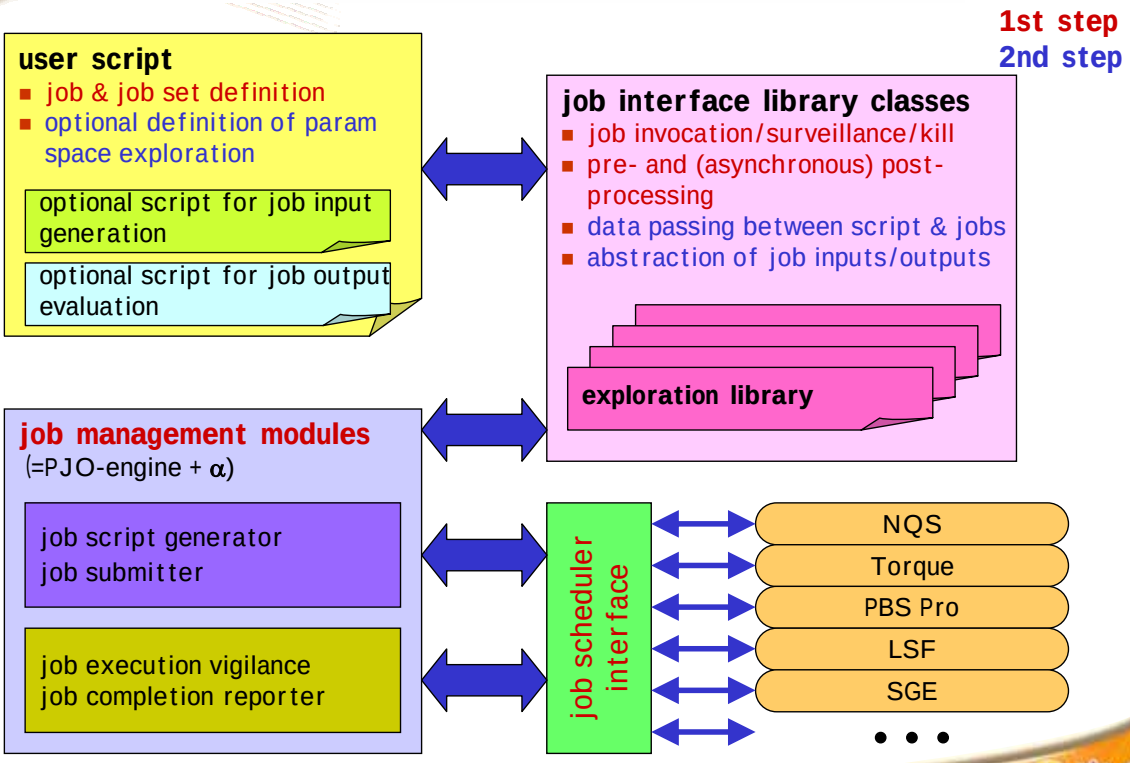
HP&P Script Language

# Design Concept

- Must not be MS-Word
  - easy-to-write does not mean nothing-to-write.
- Must not be TEX
  - rich functionality does not mean do-it-yourself for everything.
- So LATEX in some sense
  - reasonably easy-to-write and reasonably customizable.
  - encourages style-file wizards with powerful built-ins and well-designed standard interfaces.

## HP&P Script Language

# System Architecture






## HP&P Script Language

# How It Looks Like Now

```
package example;
use restrict;                  # module to control job concurrency
use parallel;                  # module to execute jobs in parallel
$myjobs=example->new();
$myjobs->{Jobset_exec}=        "pathname of program executable";
$myjobs->{Jobset_args}=        "printf string for arguments" or
                               &function_to_create_arg_string;
$myjobs->{Jobset_before}=      optional pre-job handler*;
$myjobs->{Jobset_after}=       optional post-job handler*;
$myjobs->{Par_njob}=           number of jobs to submit;
$myjobs->{Par_after_all}=      optional finalize handler*;
$myjobs->{Restrict_max}=       number of jobs to run concurrently;
$myjobs->start();              # start jobs
```

*handler : "string to eval" or &function_to_invoke

still OO & busy
immature yet

## HP&P Script Language

# How It Can Look Like

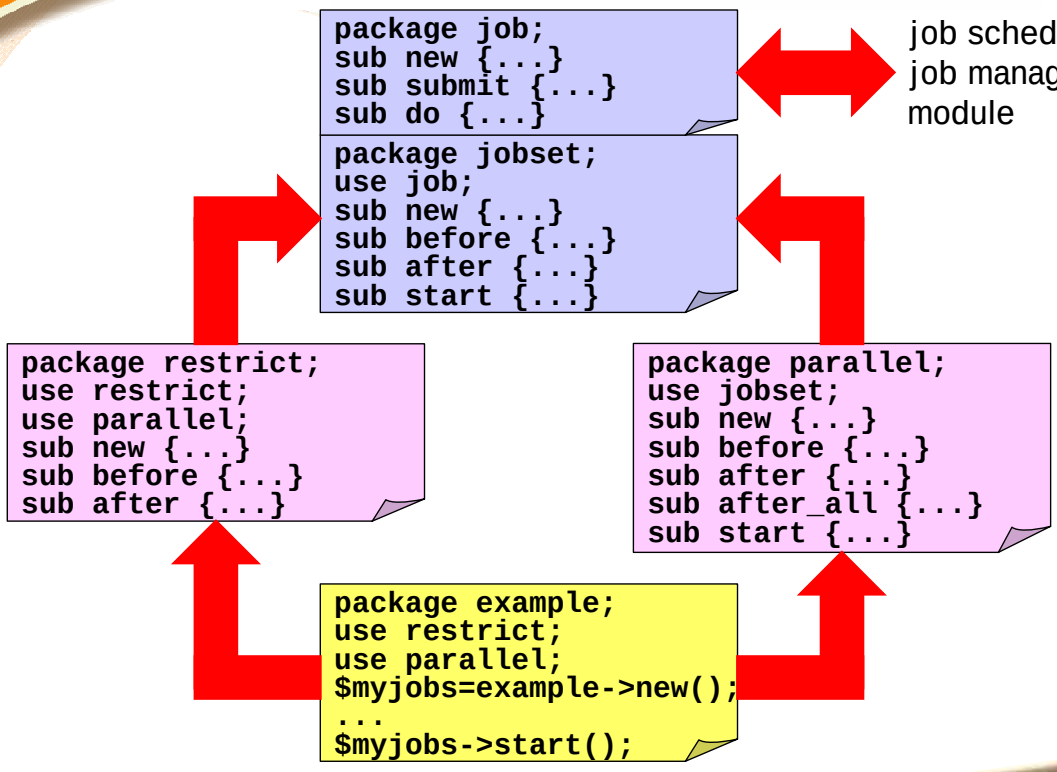
```
package example;
use restrict;                    #  module to control job concurrency
use parallel;                    #  module to execute jobs in parallel
example->start(
  exec=>       "pathname of program executable",
  args=>       "printf string for arguments"  or
               &function_to_create_arg_string,
  before=>     optional pre-job handler*,
  after=>      optional post-job handler*,
  njob=>       number of jobs to submit,
  after_all=>  optional finalize handler*,
  max=>        number of  jobs to run concurrently,
);
```

*handler : "string to eval" or &function_to_invoke



## HP&P Script Language

# How Built Now



HP&P Script Language

# Next Step: I/O Interface

For a Parameter-Sweep of a Simulation

- What they are doing
  - edit input file for each job
  - submit each job manually with the input file for it
  - view output file to get what they want
- What I am doing = write perl scripts ...
  - to create job-specific input from template
  - to do qsub with various input parameters
  - to extract what I want to make CSV
- What we have to do
  = find easiest application-specific way ...
  - to create input, to submit job, to examine output
  - with smallest effort to learn CS sorcery
  - with most sophisticated tech of CS warlock



Concluding Remark

# Let's Discuss Issues on ...

- Functionality v.s. Simplicity
  - How do we design a language gradually leading programmers from "hello world" to the level at which they feel satisfaction?
- Learning v.s. Teaching
  - How do we design a language which programmers easily learn and/or designers easily teach to them?
- Can you give a good name to our script language system?

We got, Xcrypt, from Costin.

T2K Open Supercomputer Alliance