

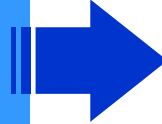


HPCS Productivity Overview

Dr. Jeremy Kepner / MIT Lincoln Laboratory

This work is sponsored by the Defense Advanced Research Projects Administration under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

- **Introduction**



- *HPCS Goals*
- *Productivity Challenges*
- *Productivity Team Organization*

- Workflows

- Benchmarks

- Productivity Formula

- Summary



HPCS Focus



- **Unique combined focus from the beginning on**
 - **Designing petascale systems for a broad range of missions**
 - **Improving the usability of such systems**
- **Developing a methodology for measuring these improvements is the focus of the Productivity Team**



HPCS Phase II Teams



➤ Create a new generation of **economically viable computing systems** and a **procurement methodology** for the security/industrial community (2010)

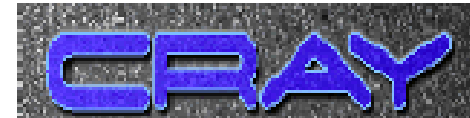
Industry



PI: Elnozahy



PI: Mitchell



PI: Smith

Mission Partners



Office of Science
U.S. Department of Energy



Productivity Team (Lincoln Lead)



PI: Kepner



PI: Lucas



PI: Basili



PI: Benson & Snaveley



PI: Dongarra



PI: Sroka

PIs: Vetter, Lusk, Post, Bailey

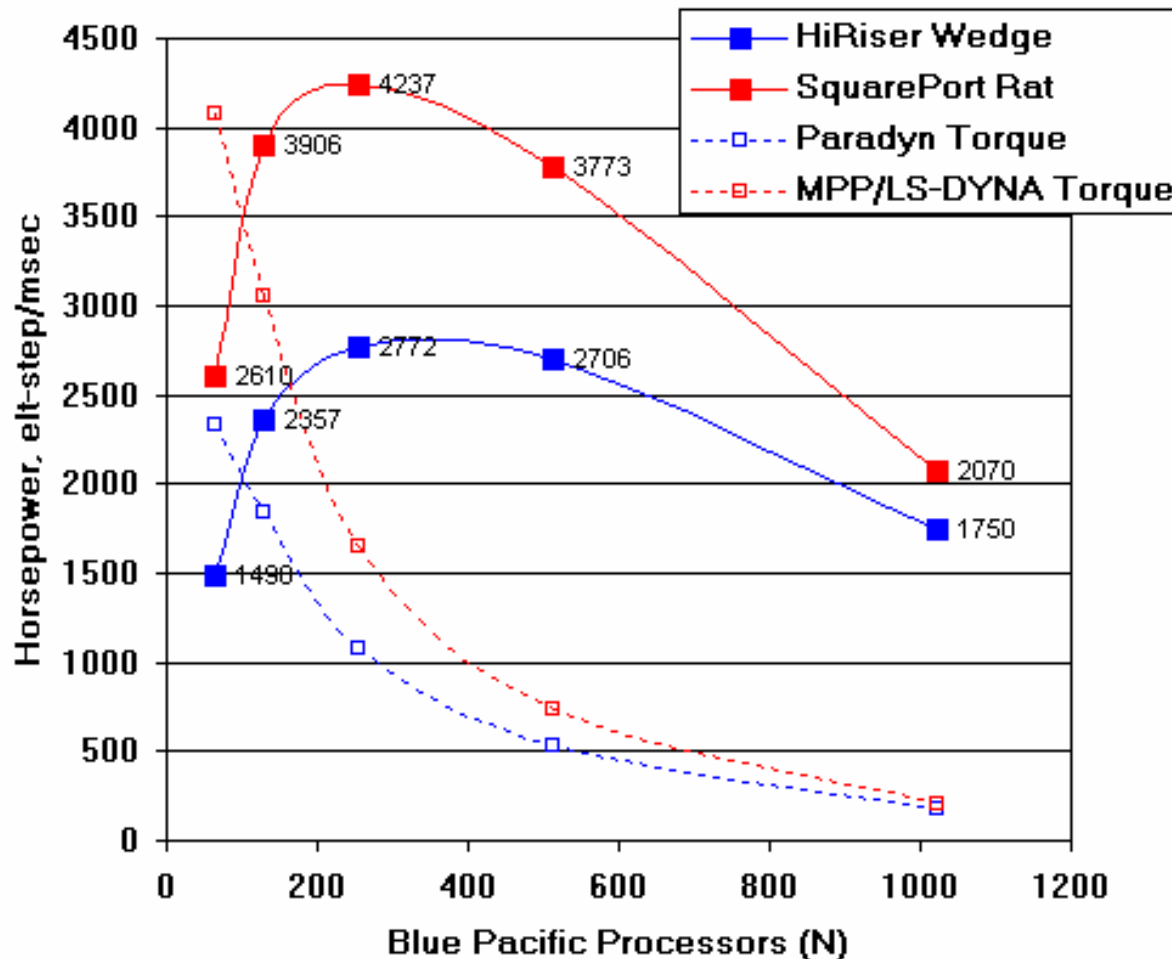
PIs: Gilbert, Edelman, Ahalt, Mitchell



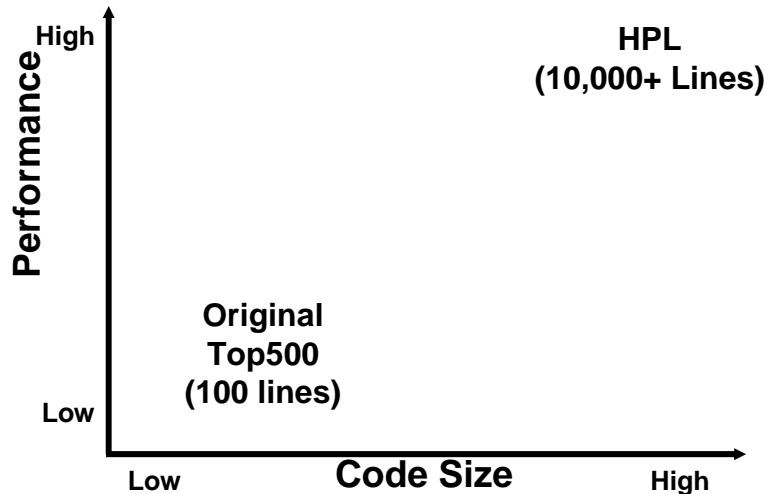
Performance



- What limits current mission applications?
 - Interconnect, memory subsystem, compilers???



Simplicity vs Performance



- Does performance require lots of coding?
- Are there better HPC programming environments?
 - Higher Level Programming Languages
 - Higher Level Parallel Programming Models
- Portability?
 - Must run legacy well

RandomAccess

• MPI Code

```

ran = starts ((NUPDATE/PROCS) * myproc);

for (i=0; i<PROCS; i++)
    pe_update_done[i] = FALSE;

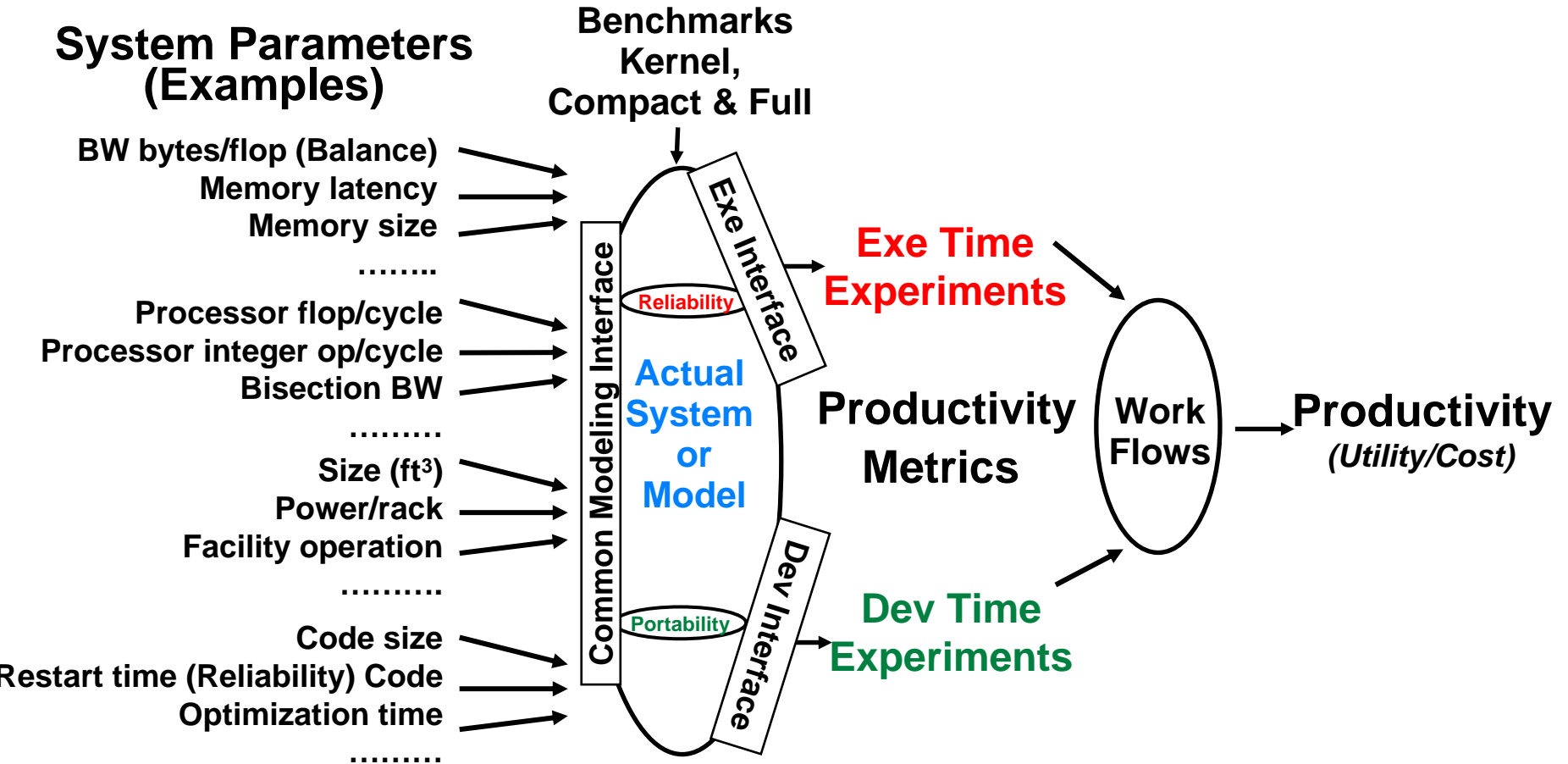
send_cnt = NUPDATE/PROCS;

while (lc_recv_done == FALSE) {
    if (send_cnt > 0) {
        /* Initialize local buckets */
        for (i=0; i<PROCS; i++) {
            for (l=0; l<EXTBK; l++) {
                lc_buck[l][i][SLOT_CNT] = FIRST_SLOT;
                lc_buck[l][i][DONE] = FALSE;
            }
        }
        /* Fill local buckets until one is full or out of data */
        next_slot = FIRST_SLOT;
        while (next_slot < BK_SZ+2 && send_cnt > 0) {
            ran = (ran << 1) ^ ((int64) ran < 0 ? POLY : 0);
            which_pe = (ran >> (LTABSIZ - LPROCS)) & (PROCS - 1);
            sub_buk = (ran >> (LTABSIZ - (LPROCS + LEXTBK)) & (EXTBK - 1));
            next_slot = lc_buck[which_pe][sub_buk][SLOT_CNT];
            lc_buck[which_pe][sub_buk][next_slot] = ran;
            lc_buck[which_pe][sub_buk][SLOT_CNT] = ++next_slot;
            send_cnt--;
        }
    }
    if (send_cnt == 0)
        for (l=0; l<PROCS; l++)
            for (i=0; i<EXTBK; i++)
                lc_buck[l][i][DONE] = TRUE;
    /* Compress the buckets */
    for (k=0; k<PROCS; k++) {
        /* For each destination compress the EXTBK buckets
           bucket for sending */
        for (l=1; l<EXTBK; l++) {
            /* Find the size of the gap between buckets */
            gap = (l * BK_SZ + 2) - lc_buck[k][0][SLOT_CNT];
            /* Find the amount of data in the bucket to be moved */
            bsz = lc_buck[k][l][SLOT_CNT] - 2;
            /* Calculate the amount of data that needs to be moved to
               have a single bucket */
            mindata = gap < bsz ? gap : bsz;
            /* Calculate where the data is coming from */
            tail = l * (BK_SZ + 2) + lc_buck[k][l][SLOT_CNT] - 1;
            /* Calculate where the data is going to */
            next_slot = lc_buck[k][0][SLOT_CNT];
            /* Move the data */
            for (m=tail; m>=tail-mindata; m--) {
                lc_buck[k][0][next_slot] = lc_buck[k][l][m];
                next_slot++;
            }
            lc_buck[k][l][SLOT_CNT] = next_slot;
            if (gap < bsz)
                lc_buck[k][0][SLOT_CNT] += bsz - gap;
        }
    }
    /* End of compress loop */
    #endif
} /* End of sending loop */
    
```

• UPC

```

for (j=0; j<128; j++)
    ran[j] = starts ((NUPDATE/128) * j);
for (i=0; i<NUPDATE/128; i++) {
#pragma ivdep
    for (j=0; j<128; j++) {
        ran[j] = (ran[j] << 1) ^ ((int64)
            ran[j] < 0 ? POLY : 0);
        table[ran[j] & (TABSIZE-1)] ^= stable[ran[j]
            >> (64-LSTSIZE)];
    }
}
    
```

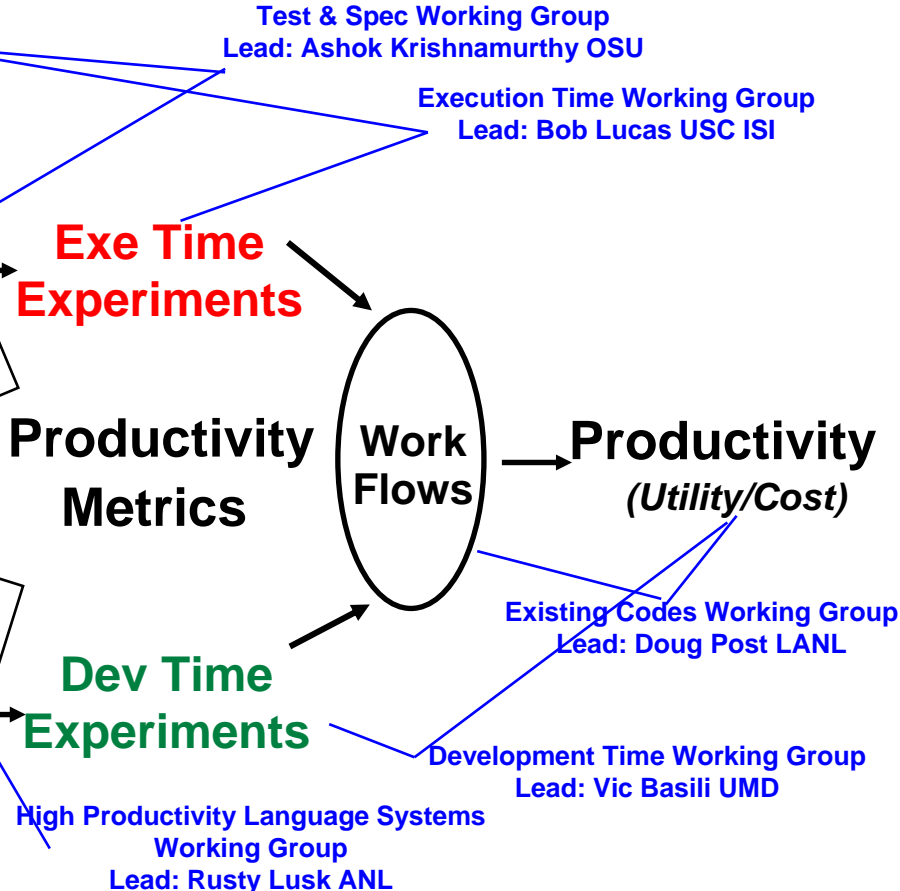
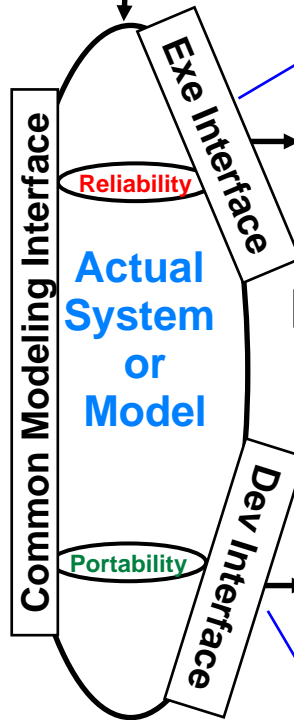


- Captures major elements that go into evaluating a system
- Builds on current HPC acquisition processes

System Parameters (Examples)

- BW bytes/flop (Balance)
- Memory latency
- Memory size
-
- Processor flop/cycle
- Processor integer op/cycle
- Bisection BW
-
- Size (ft³)
- Power/rack
- Facility operation
-
- Code size
- Restart time (Reliability) Code
- Optimization time
-

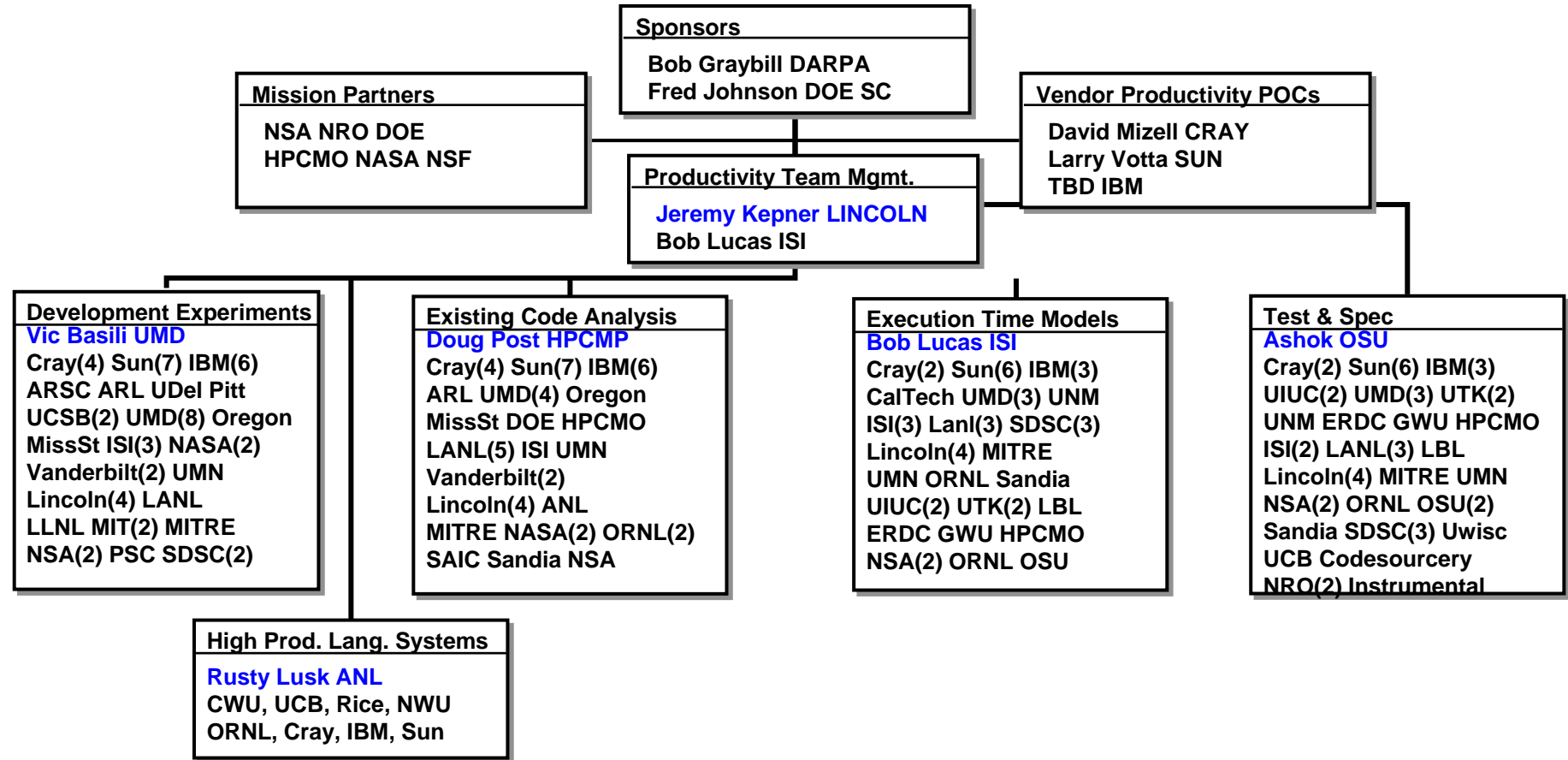
Benchmarks
Kernel,
Compact & Full



- Multi-organizational team that spans the HPC community
- Strong contingent from the software engineering community

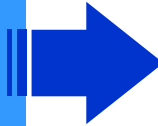


Productivity Team



- Introduction

- **Workflows**

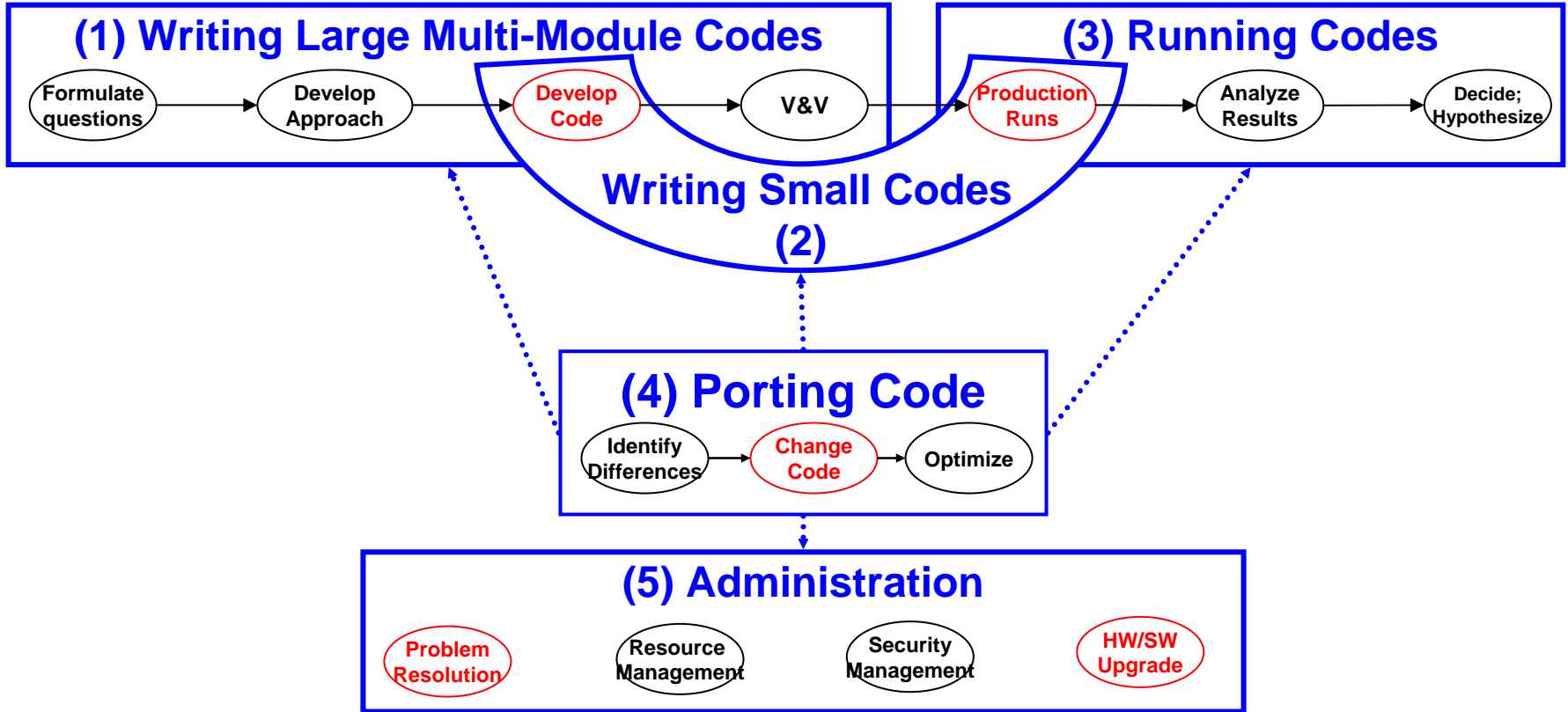


- *Level 1 and 2 workflows*
- *Workflow Usage*
- *Markov Models*

- Benchmarks

- Productivity Formula

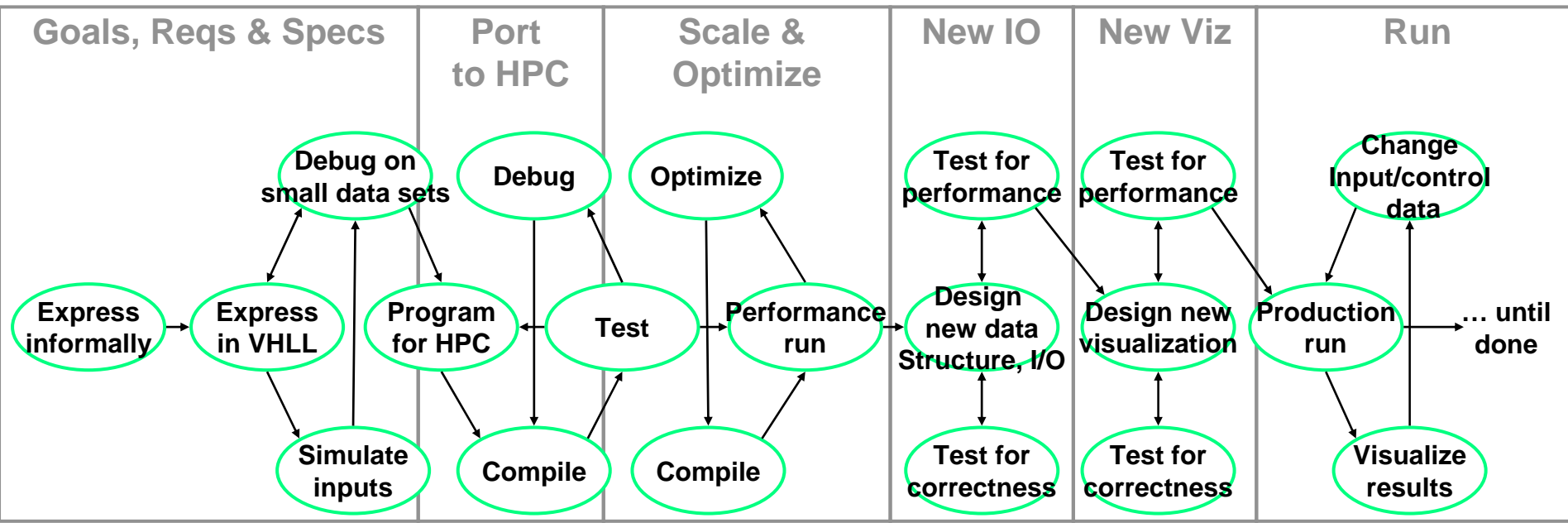
- Summary



- Workflows comprise many steps; many overlapping
- Item in **red** represent areas with highest HPC specific interest



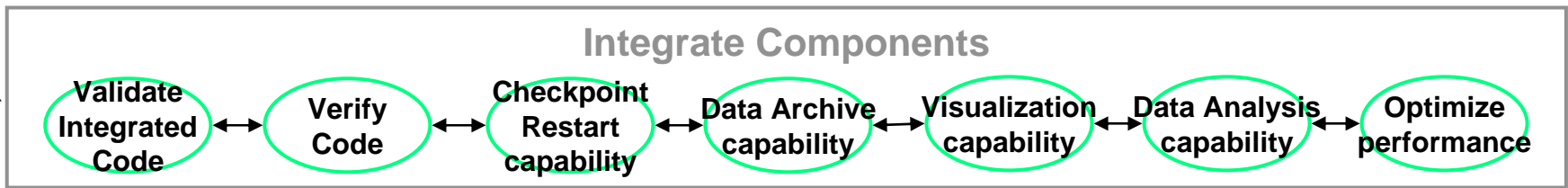
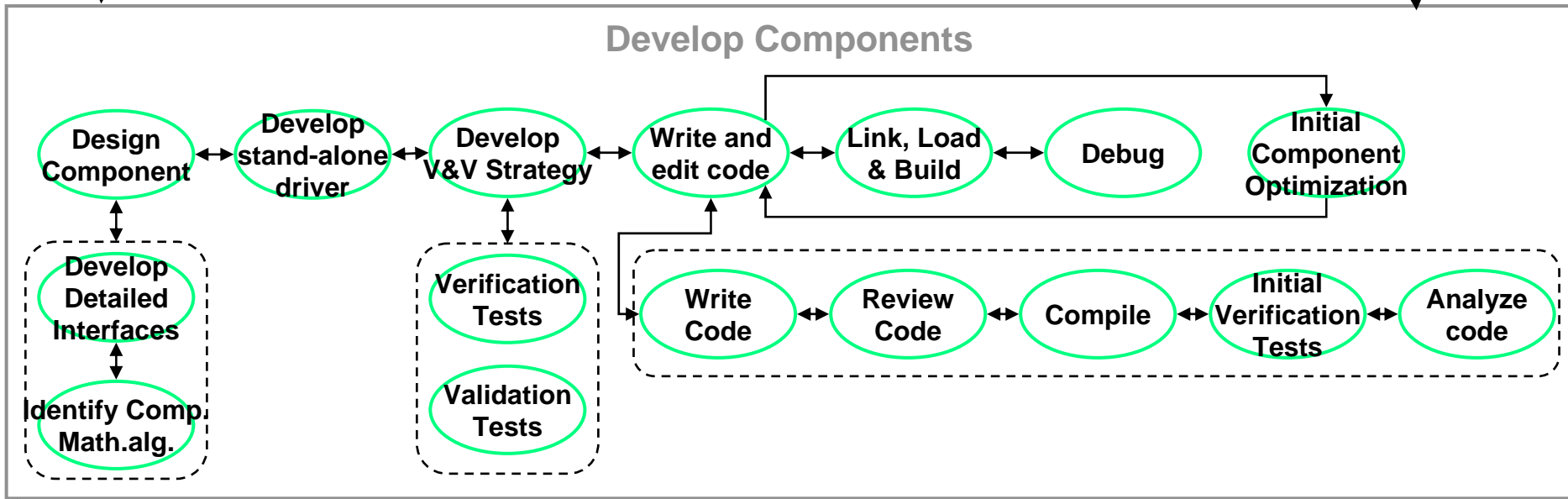
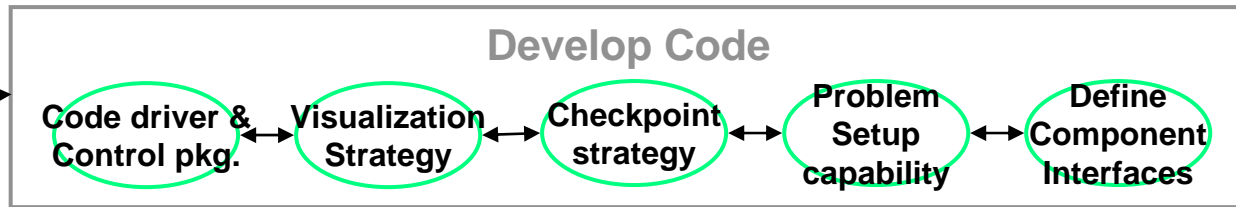
Level 2: Coding small programs quickly

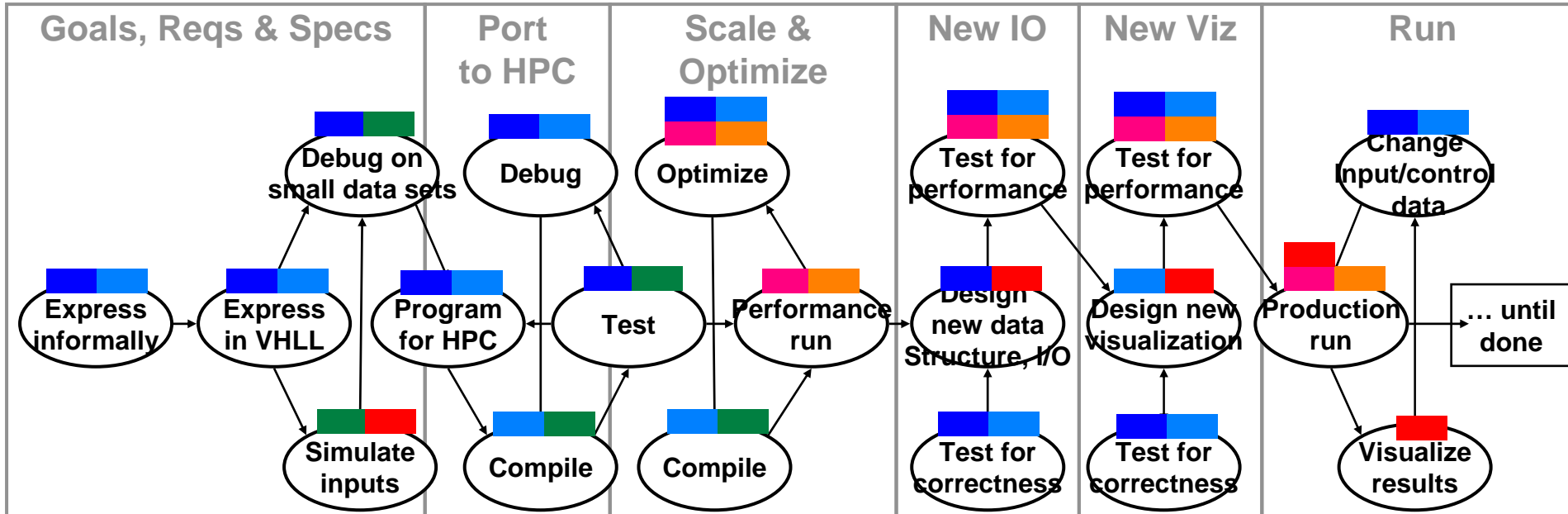


- Allows vendors to specifically identify which steps they are addressing with their technologies



Multi-Module Development

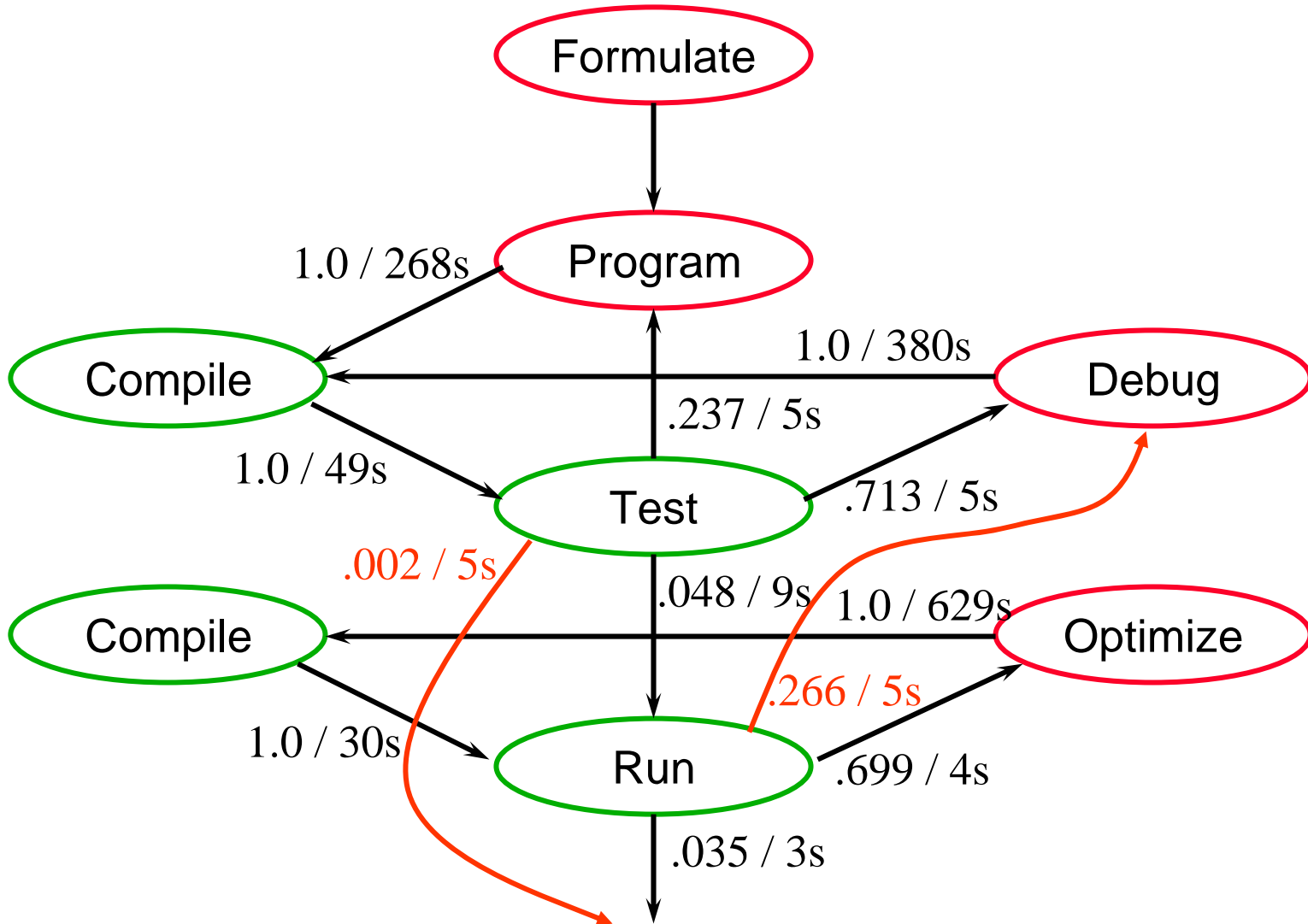




- Compare proposed technologies against workflow steps
- Predict and then measure impact of technology

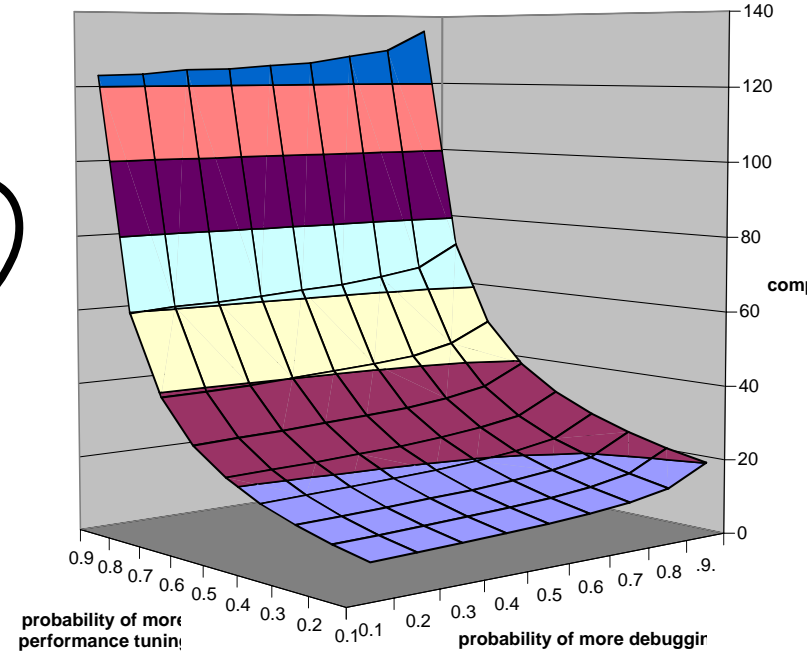
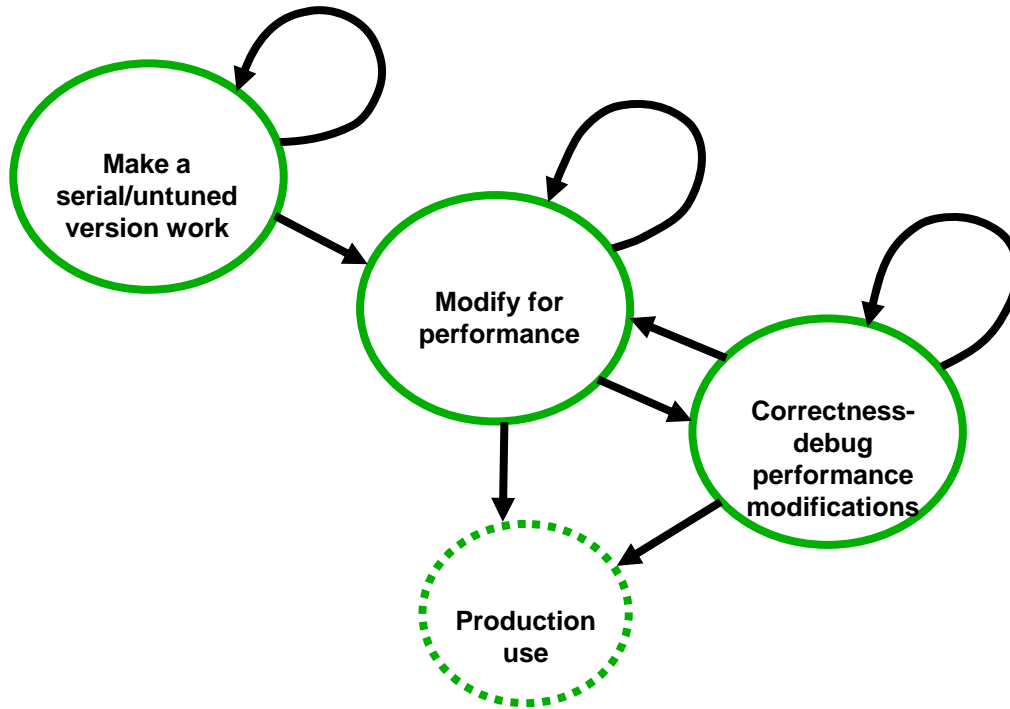


Small Code Level 2 Work Flow Example Markov Model - Classroom (UCSB) Data

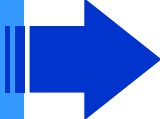


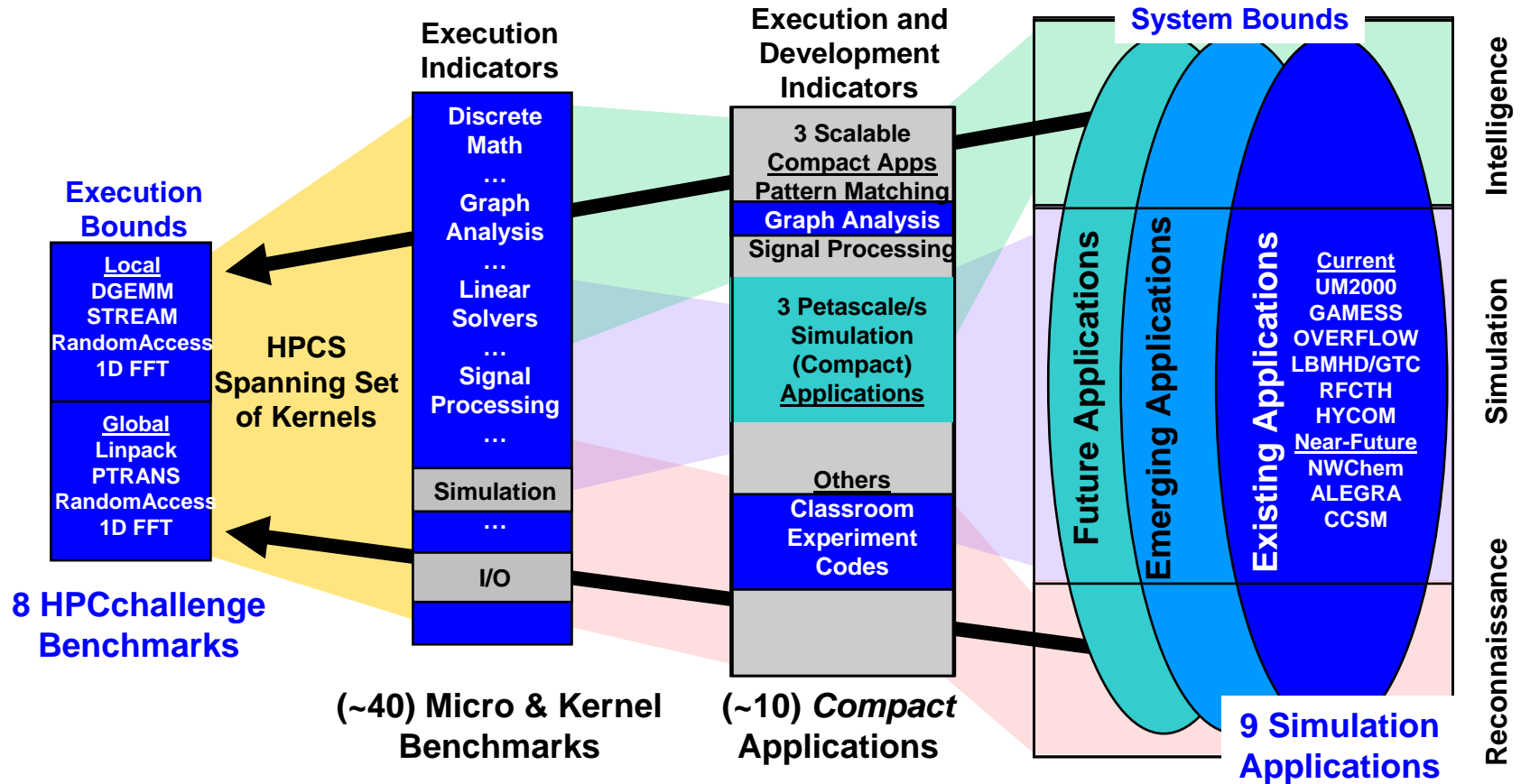
Revised Porting Code Workflow

Workflow Sensitivity Analysis



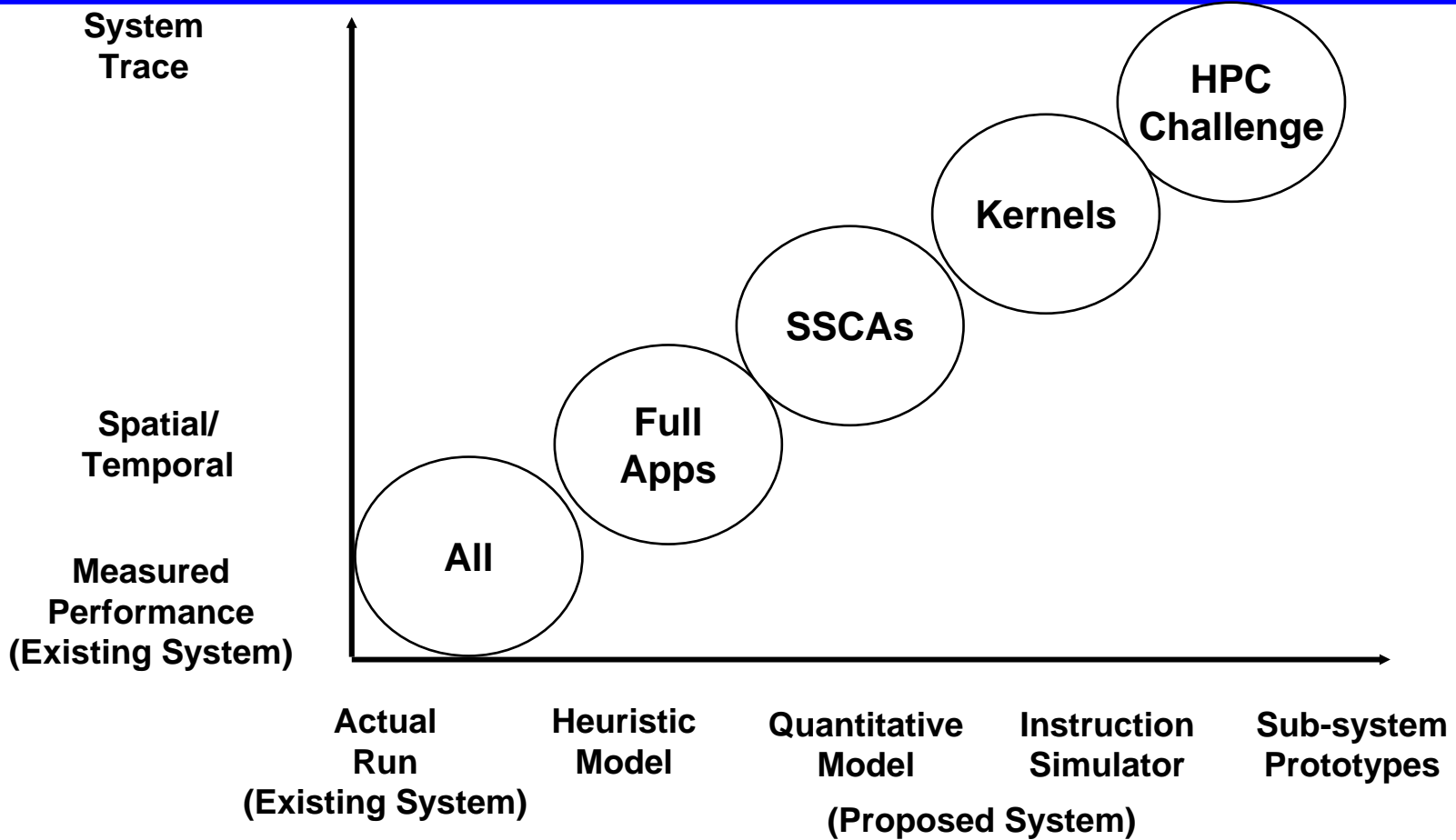
Slide: Courtesy of Cray

- Introduction
- Workflows
- **Benchmarks** 
- *Execution Time*
- *Development Time*
- Productivity Formula
- Summary

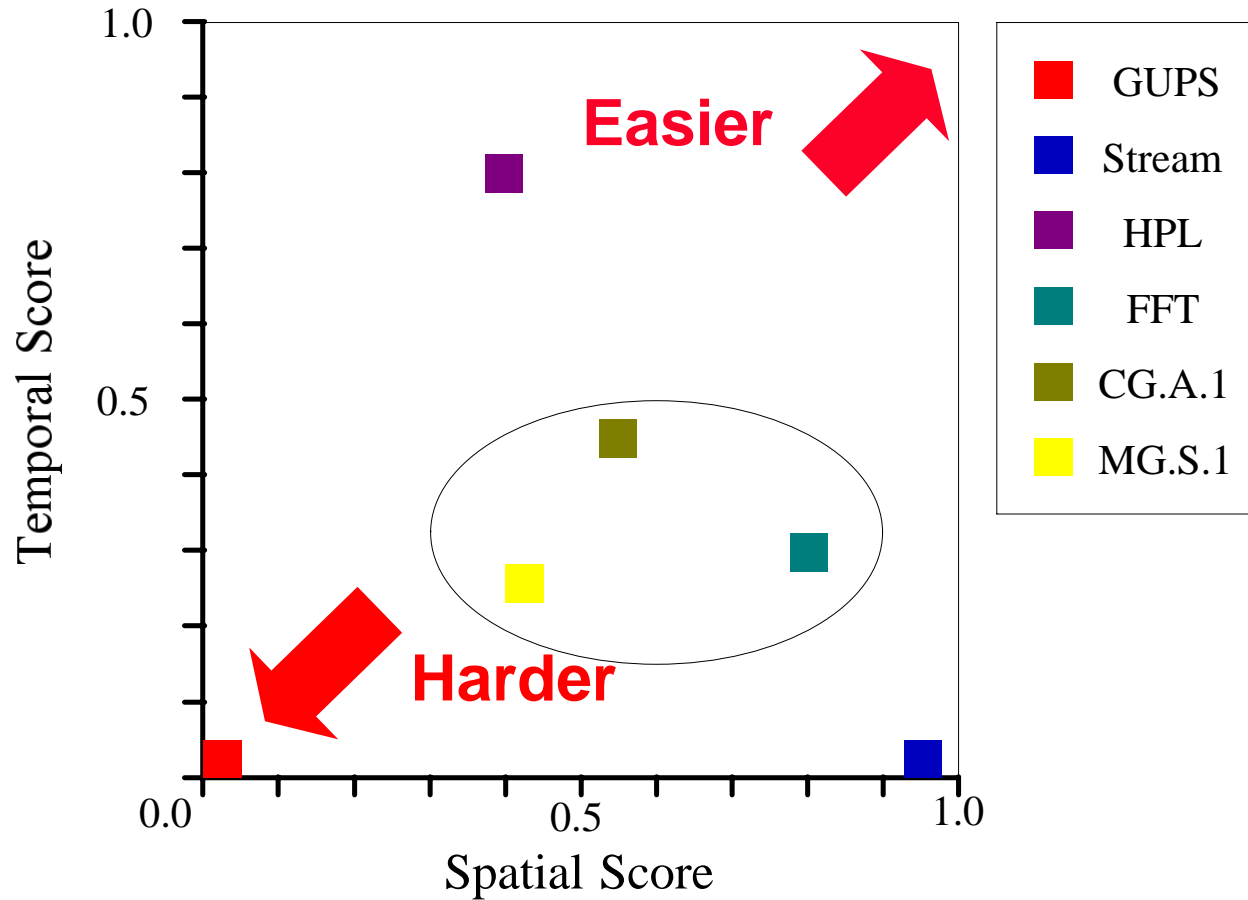


Spectrum of benchmarks provide different views of system

- HPCchallenge pushes spatial and temporal boundaries; sets performance bounds
- Applications drive system issues; set legacy code performance bounds
- Kernels and Compact Apps for deeper analysis of execution and development time

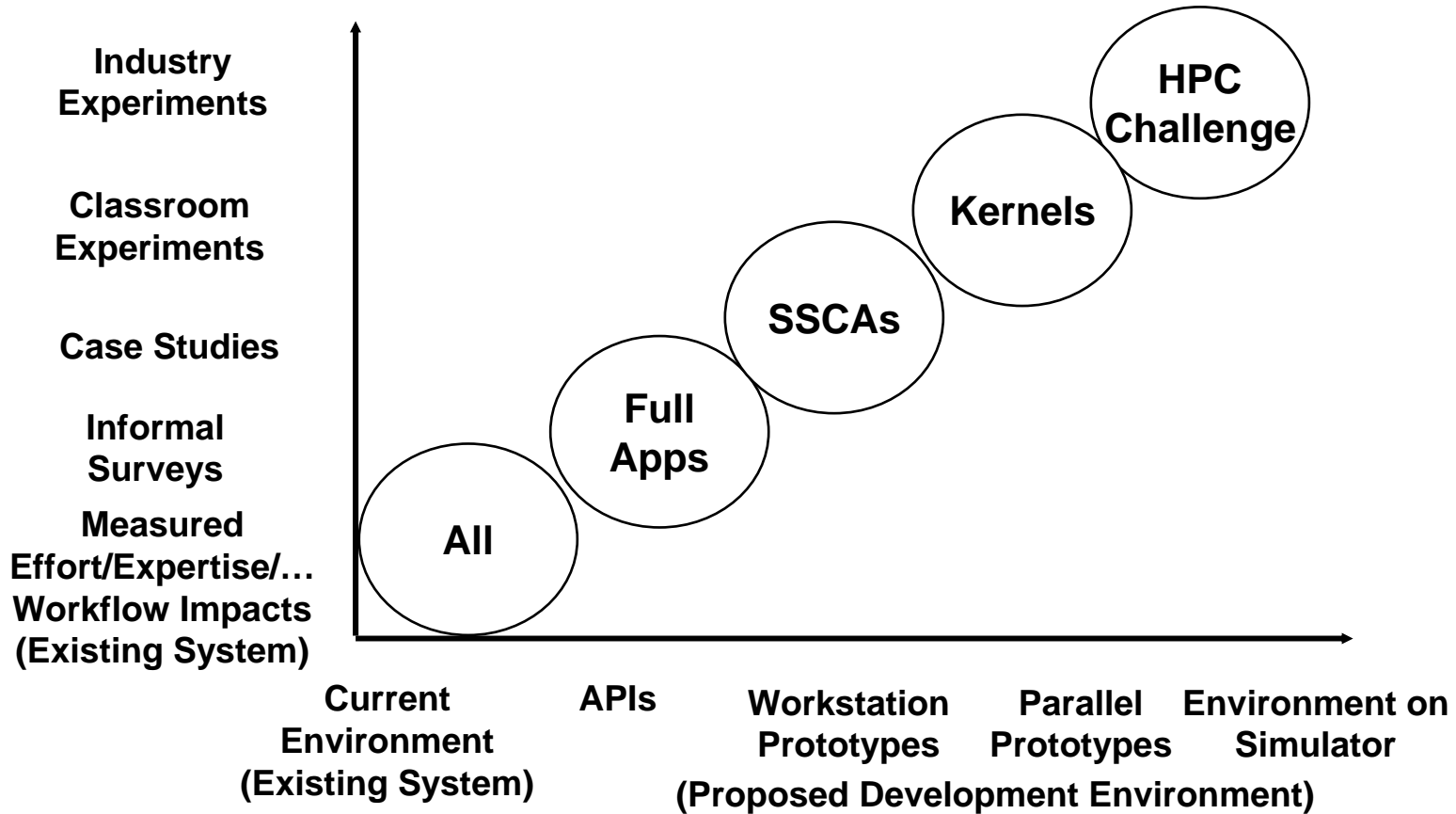


- HPCS Benchmarks provide a standard context for evaluating performance of a system
- Can be applied at different stages of technology maturity





Benchmarks and Programmer Effort



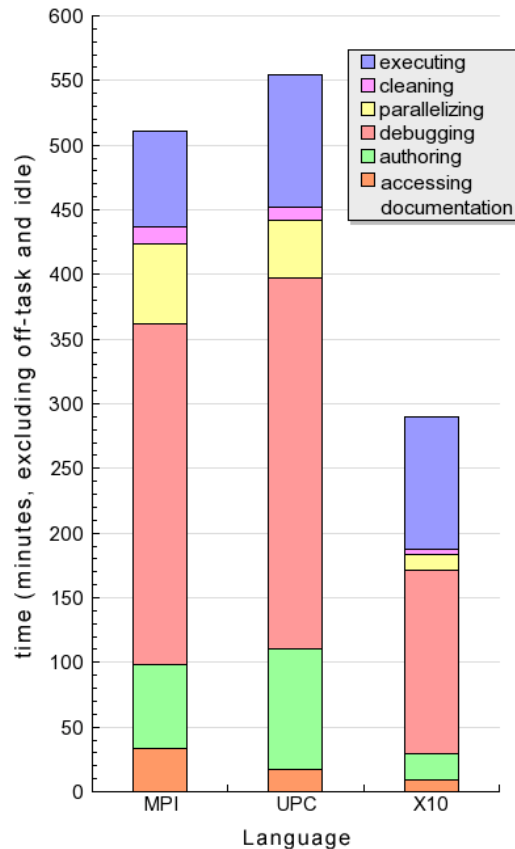
- HPCS Benchmarks provide a standard context for evaluating development environment a system
- Can be applied at different stages of technology maturity



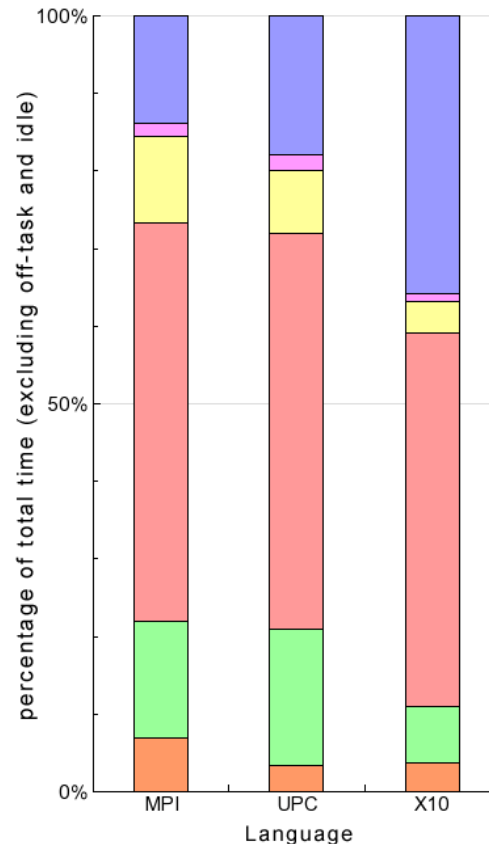
Example: Average Development Time by Language (PSC team)



Absolute Time



Percentage of Total



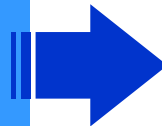
Comparing average development times between languages, several observations are clear:

- Average development time for subjects using X10 was significantly lower than that for subjects using UPC and MPI.
- The relative time debugging was approximately the same for all languages.
- X10 programmers spent relatively more time executing code and relatively less time authoring and tidying code.
- Subjects using MPI spent more time accessing documentation (tutorials were online; more documentation is available).
- For this experiment, the effect of batch vs. interactive environments was probably significant.

Slide: Courtesy of IBM

- Introduction
- Workflows
- Benchmarks

- **Productivity Formula**



- *Utility/Cost*
- *Future Vision*

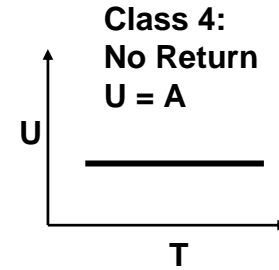
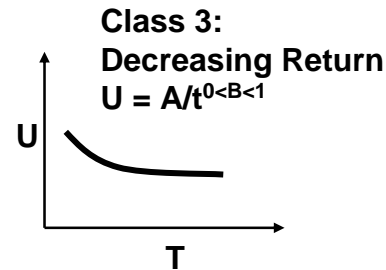
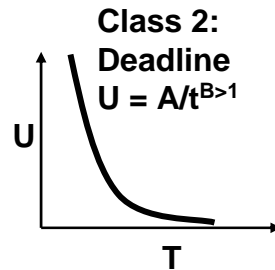
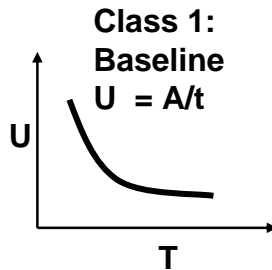
- Summary

$$\Psi \equiv \frac{U}{C} = \frac{U(T)}{C_S + C_O + C_M}$$

ψ = productivity [utility/\$]
 U = utility [user specified]
 T = time to solution [time]
 C = total cost [\$]

C_S = software cost [\$]
 C_O = operation cost [\$]
 C_M = machine cost [\$]

- **Utility is value user places on getting a result at time T**
 - Assume $U(T) = A/T^B$ (Biegel)



- **Software costs include time spent by users developing their codes**
- **Operating costs include admin time, electric and building costs**
- **Productivity formula is tailored by each user through use of functional work flows**
 - Developing Large multi-module codes
 - Developing Small Codes
 - Running applications
 - Porting codes
 - Administration

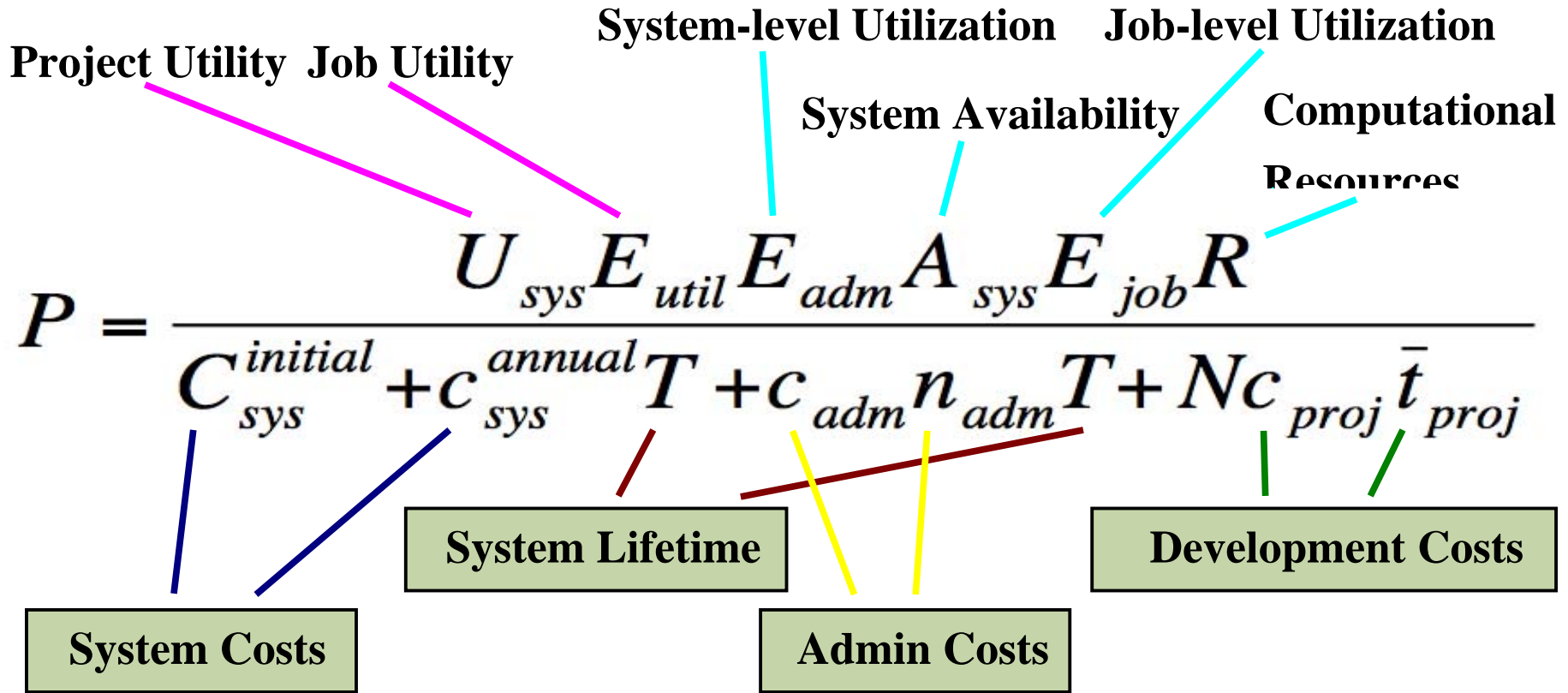


Example: Sun Figure of Merit

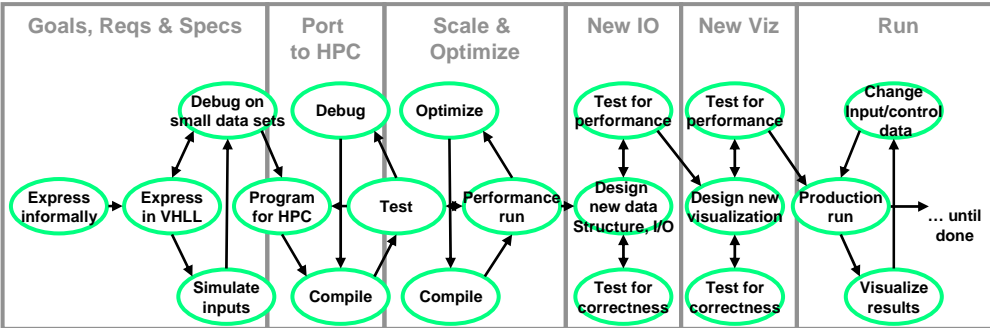


Subjective Values:
Site-specific

Objective Measurable
System Properties



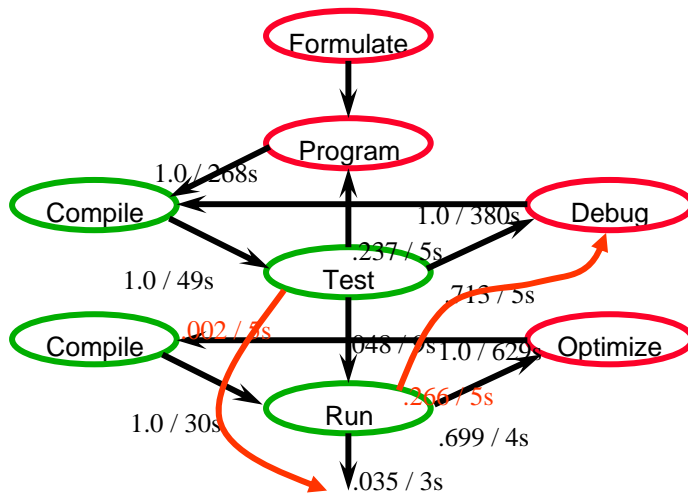
Workflows from Case Studies



- Use Markov models to estimate user effort (C_S)
 - Also provides info on achieved performance (U)
 - Admin workflow can be used to estimate admin effort (C_S)

Measured activities from experiments

Time estimates from user surveys



Achieved Performance

$$\Psi \equiv \frac{U(T)}{C_S + C_O + C_M}$$

Estimated User Effort

(Admin Workflow)

Markov Models from Experiments

Productivity Formula



Summary



- **HPCS Goals**
 - Provide a new generation of economically viable high productivity computing systems for the national security and industrial user community (2010)
- **Impact**
 - Performance (time-to-solution): speedup critical national security applications by a factor of 10X to 40X
 - Programmability (idea-to-first-solution): reduce cost and time of developing application solutions
 - Portability (transparency): insulate research and operational application software from system
 - Robustness (reliability): apply all known techniques to protect against outside attacks, hardware faults, & programming errors
- **Vendors have developed a variety of hardware and software technologies with significant potential**
- **HPSS Productivity Team goal is to develop an acquisition quality framework for HPC systems that includes**
 - Development time
 - Execution time