What do we mean by scalability?

- **Qualitative:** Will this software work well for substantially larger problems?

- **Semi-quantitative:** Can you solve a problem *n* times as large in the same amount of time if you have *n* times as many resources?
  - **What if you had a CPU *n* times faster?**
  - **What if you had *n* CPUs?**

  A dual of this question is:

  would a problem *n* times larger take at most *n* times longer with no change in resources?

What is a problem *n* times larger?

- Problem Dimensions
  - **amount of data**
    - number of primary objects
    - size of primary objects
  - **number of concurrent users**
- Other
  - **number of kinds of data**
  - **number of primary object components, or additional component complexity**
  - **...**

What are *n* times the resources?

- CPU?
- Disk space?
- Memory?
- Network bandwidth?

A pragmatic approach is “*n* times the budget.”

This usually means parallelism (or replication, or other concurrency).

Scalability in practice?

- In practice, scalability may be addressed, not for arbitrary *n*, but for reasonable ranges.
- I propose we ask the question in terms of problem dimensions and scales we expected when we wrote the proposal:

  10^9 primary objects, 1 MB/primary object, 100’s of concurrent users

- When people ask whether a system is scalable to a particular size, they want to know whether it was engineered to handle problems of this size (this means more than "will it break?"), and whether it can deliver "reasonable" performance at this scale on a "reasonable" budget.
- System architects try to determine (predict?) what performance would be as a function of hardware resources; users decide whether that’s "reasonable."

Common scalability issues

- Algorithmic complexity
  - **worse-than-linear algorithms do not scale well: a 100-city traveling salesman problem is not 10 times harder than a 10-city one; it's 100x99x98x...x13x12x11 times harder**
  - **even algorithms linear in *n* are often painful (and sometimes unnecessary) when *n* is large**
- Reliance on singletons
- Sequentiality
- Robustness
- "Starting from zero"

Understanding scalability

- By analysis
  - overall architecture
  - individual components
- By designed experiment
  - Tomorrow’s discussion?

Scalability: overall architecture

- Where are our singletons?
  - Do they need to be singletons?
  - Current “necessary singleton” list:
    - **ONE cache (possibly distributed), and knowledge of what’s in it**
    - **ONE component that knows about all queries and their file needs**
- Sequentiality, robustness, …
- Is our intercomponent communication scalable?

Scalability: individual components

- Authors should explain
  - how their components operate
  - what would happen with *n* times more primary objects and queries
  - where their scalability concerns are (algorithmic, singletons, robustness, sequentiality, “starting from zero”, communication, other)
  - are there any worse-than-linear algorithms in use?
  - are there any steps that are linear in the number of primary objects, or in the number of queries? (These should be examined for possible improvement.)