Experiments with a Parallel Multi-Objective Evolutionary Algorithm for Scheduling

Matthew Brown
University of Southern California
Mark Johnston
Jet Propulsion Laboratory, California Institute of Technology

Outline

- NASA's Deep Space Network (DSN)
  - Overview
  - Long Range Planning & Scheduling
- Loading Analysis and Planning Software (LAPS)
  - Evolutionary Multi-Objective Algorithm
  - Parallelizing for multiple core hardware
- Results and Conclusions

The Deep Space Network (DSN)

- Current DSN comprises
  - 3 sites roughly equally spaced in longitude
  - one 70m + multiple 34m antennas at each site
- DSN supports all planetary missions + some earth orbiters + radio science/astronomy

Complex	GDSCC	CDSCC	MDSCC
Location	Goldstone, California, USA	Canberra, Australia	Madrid, Spain
Longitude	117° W	149° E	4° W
Latitude	35° N	35° S	40° N
Antennas	1 - 70m 5 - 34m	1 - 70m 2 - 34m	1 - 70m 3 - 34m
Capabilities	S, X, Ka	S, X Ka downlink only	S, X Ka downlink only

- DSN scheduling problem:

  - ~500 tracks (communications contacts) per week for ~37 DSN users, with wide variation in types of scheduling requirements
  - Goal is to have a negotiated schedule about 16 weeks ahead of realtime, and be conflict free about 8 weeks ahead
    - driven by need to sequence spacecraft well in advance

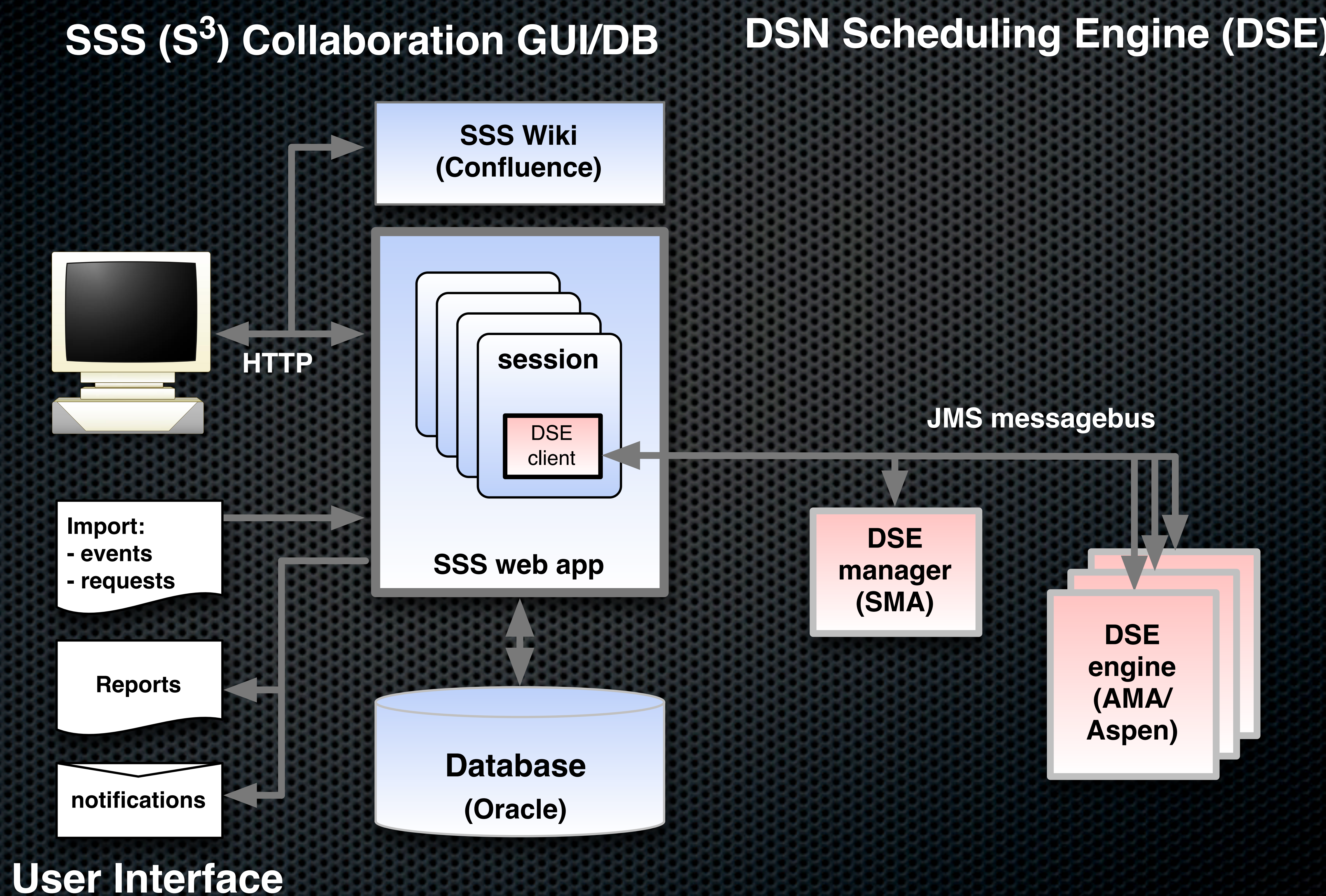
DSN Scheduling Process Phases

Process Phase	Time frame relative to execution	Software tools (software/database)	Characteristic activities
Long-range	≳ 6 months	TIGRAS (RAP version) + MADB database	• identify and resolve periods of contention • plan for extended downtime • assess proposed missions • assess long range asset options
Mid-range	few weeks out to 6 months	S^3 webapp/database	• schedule normal science operations • schedule pre-planned s/c activities (maneuvers, unique science opportunities) • generate negotiated schedules for s/c sequencing • schedule network maintenance
Near Real-time	closer than a few weeks	TIGRAS (SPS version) + Service Preparation System (SPS) database	• predict generation for execution • reschedule due to unplanned resource unavailability • respond to spacecraft emergencies • activate pre-planned launch contingencies

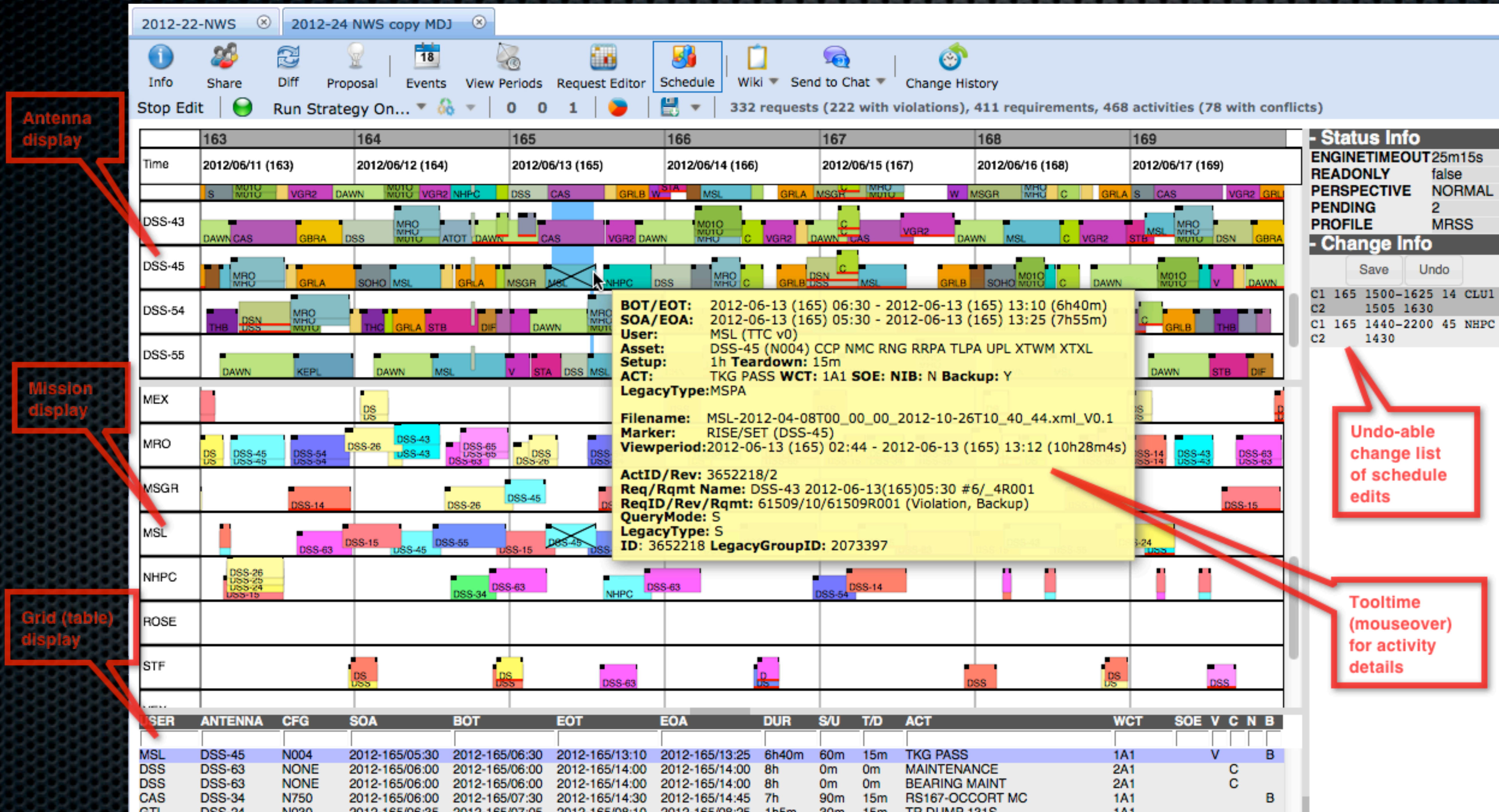
Service Scheduling Software (S^3)

- DSN has undertaken a major implementation of scheduling automation called the Service Scheduling Software (S^3) system
- Major goals are:
  - unify the scheduling software and databases into a single integrated suite covering realtime out through as much as several years into the future
  - adopt a request-driven approach to scheduling (as contrasted with the former activity-oriented scheduling)
  - develop a peer-to-peer collaboration environment for DSN users to view, edit, and negotiate schedule changes and conflict resolutions

Architectural Overview of S^3 and the DSE



Schedule Visualization and Editing



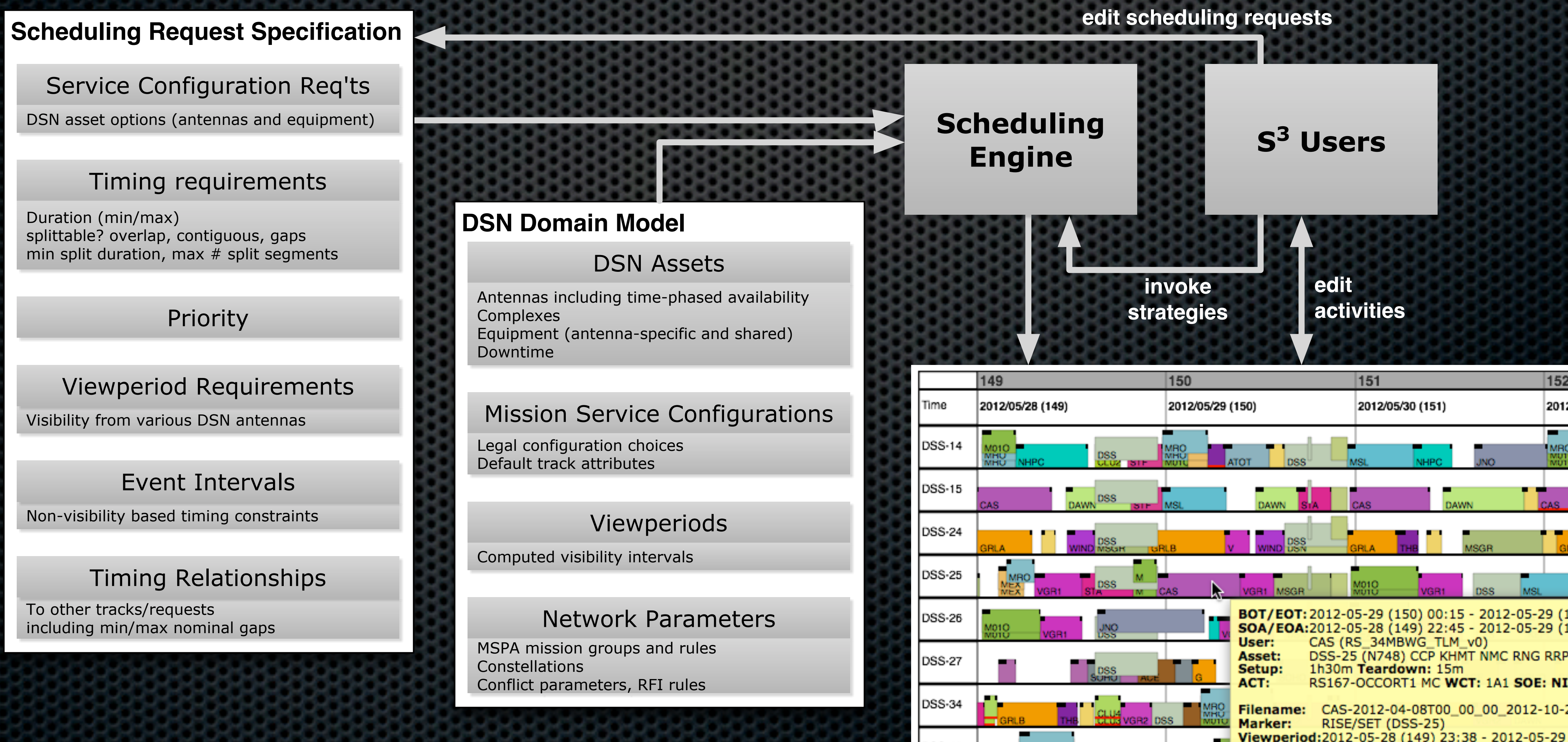
S^3 Status

- S^3 was deployed operationally in June 2011 and has been operational since that date
- About two years of DSN weekly schedules have been created and negotiated in S^3, since 2011 week 29
  - includes baseline schedules for 3 launching missions in late 2011
  - includes Mars Science Laboratory Entry/Descent/ Landing in early August 2012

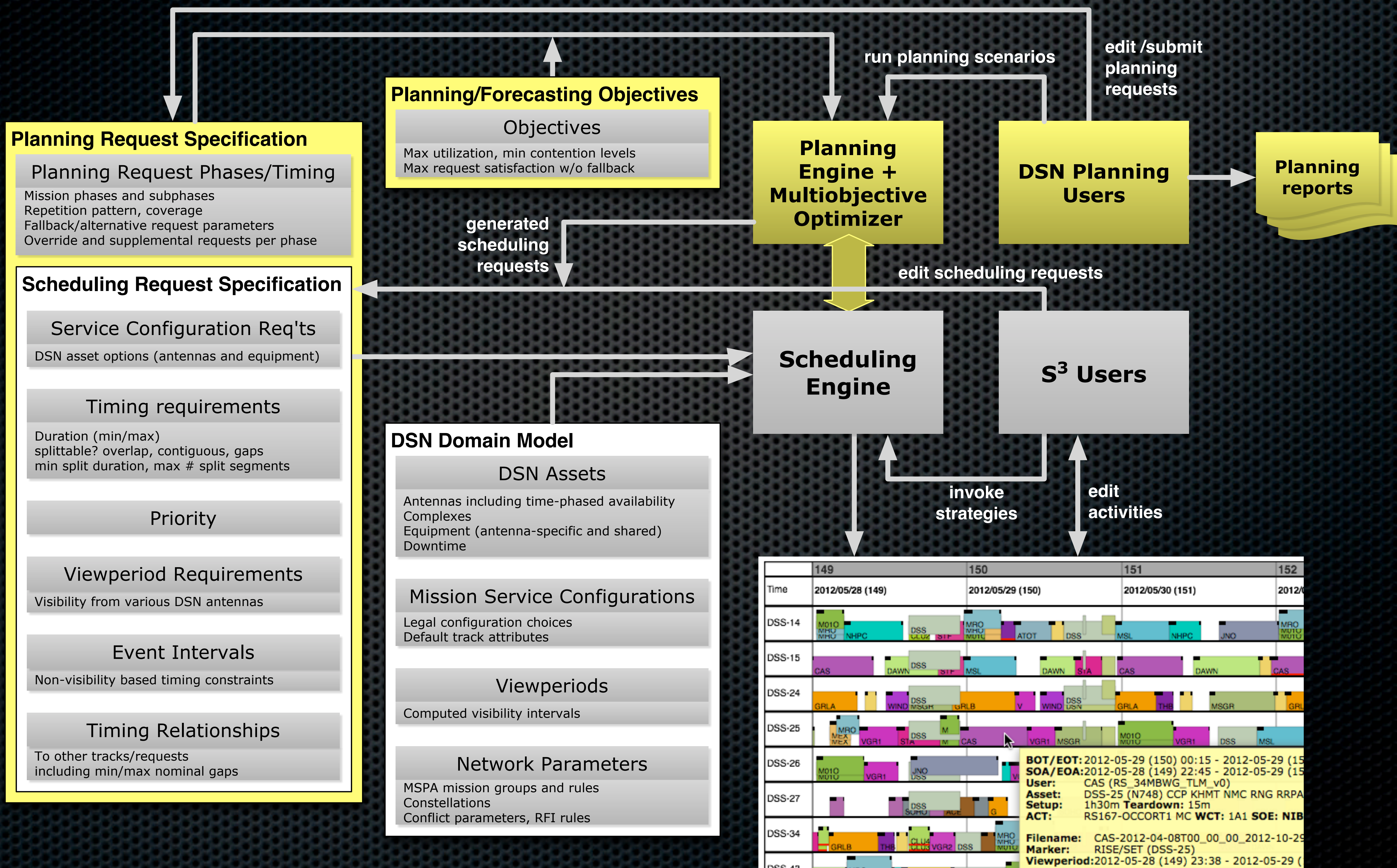
Extension of S^3 to long-range planning and forecasting

- DSN is extending S^3 functionality to long-range process
- Leverage S^3 data model and infrastructure
- Additional development is required for
  - modeling uncertainty
  - different optimization criteria
  - simplified planning request interfaces for users
  - new reporting functionality
- Optimization will explicitly use *multiobjective* algorithms to provide insight into tradeoffs among competing objectives

Extending the S^3 baseline...



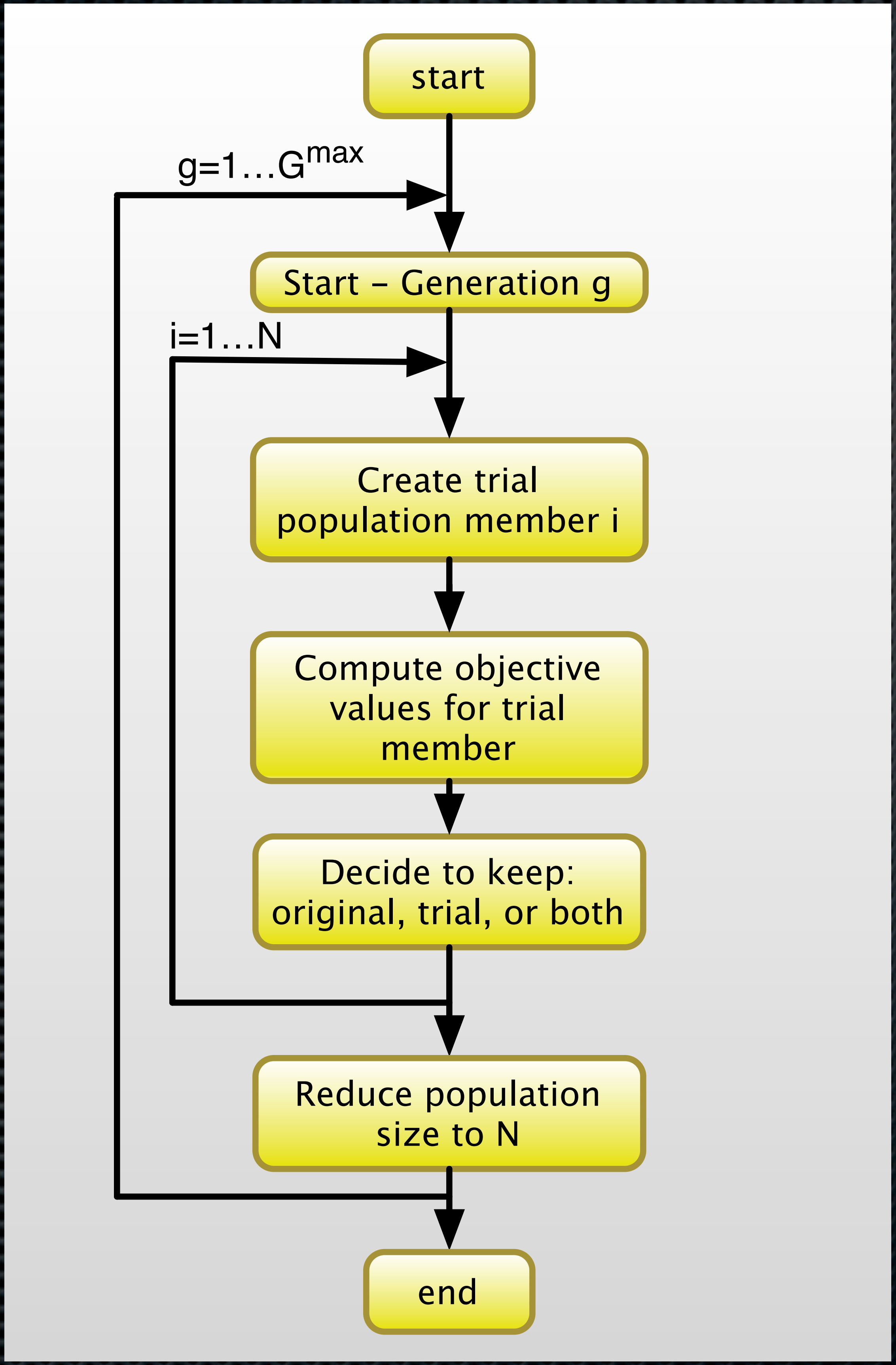
... to incorporate long-range planning functionality



Loading Analysis & Planning Software (LAPS)

- Algorithm: GDE3 (Generalized Differential Evolution 3, Kukkonen and Lampinen 2005)
  - maintains population of real-valued decision vectors
- Decision variables:
  - per time interval (nominally weekly)
    - mission relative priority
    - fallback potential (nominal, reduced, minimal)
- Objectives (minimization):
  - unscheduled requirement time (all missions)
  - total track duration scheduled on all antennas
- Sample problem: 16 weeks, all DSN missions, slightly (10%) oversubscribed

Algorithm

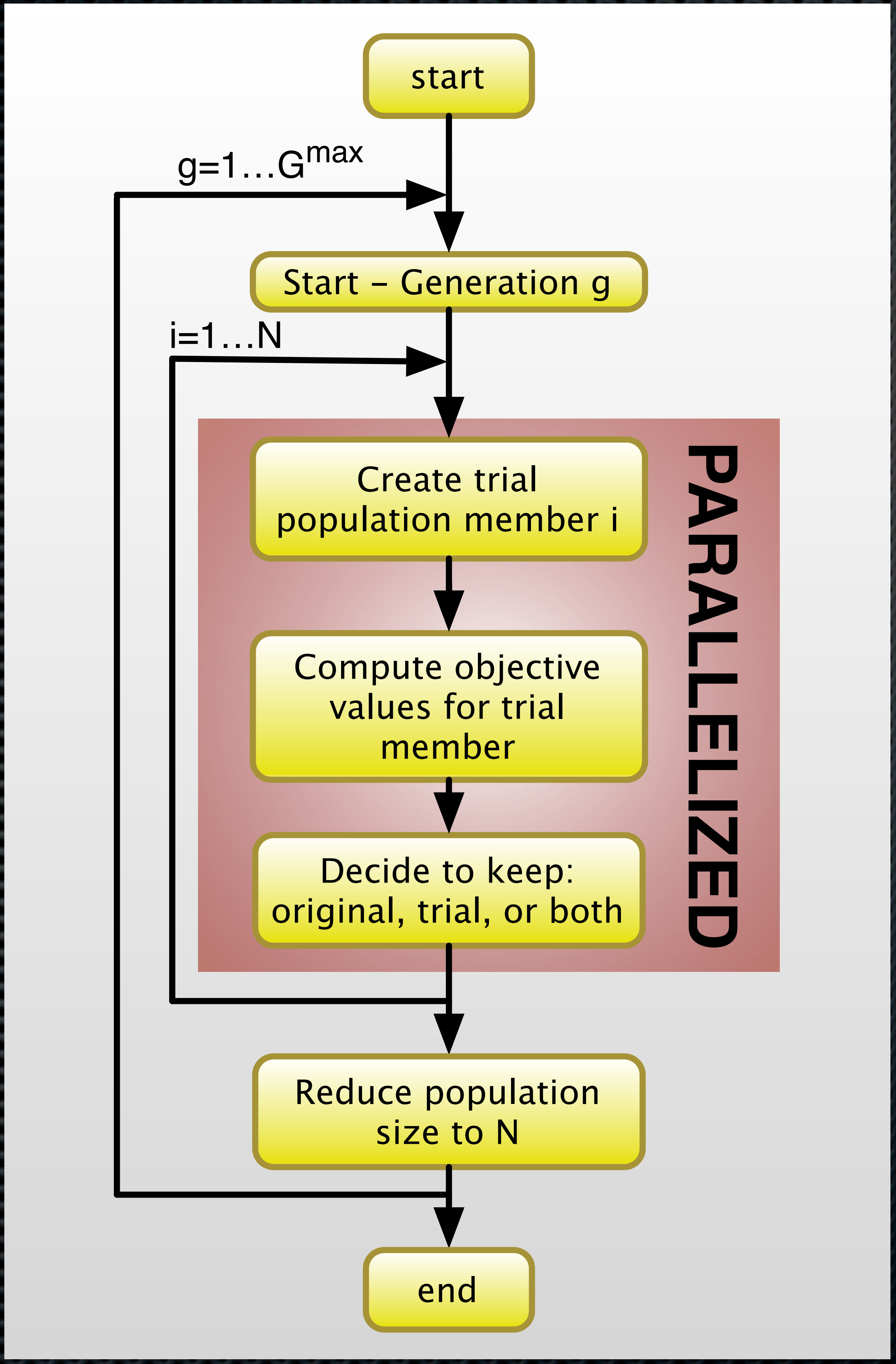


start
g=1…G^max
Start – Generation g
i=1…N
Create trial population member i
Compute objective values for trial member
Decide to keep: original, trial, or both
Reduce population size to N
end

Java 7 ForkJoin functionality

- New with Java 7 is API for easily parallelizing algorithms to use multiple cores: ForkJoin
- Applied to GDE3 as follows:
  - (Fork) For each generation, create N Java *Callable* tasks that implement offspring generation, including time-consuming the objective calculation
  - (Join) When all N tasks have completed, perform the population reduction as needed, then prepare for the next generation
- By default ForkJoin uses maximum number of cores supported by hardware

Algorithm

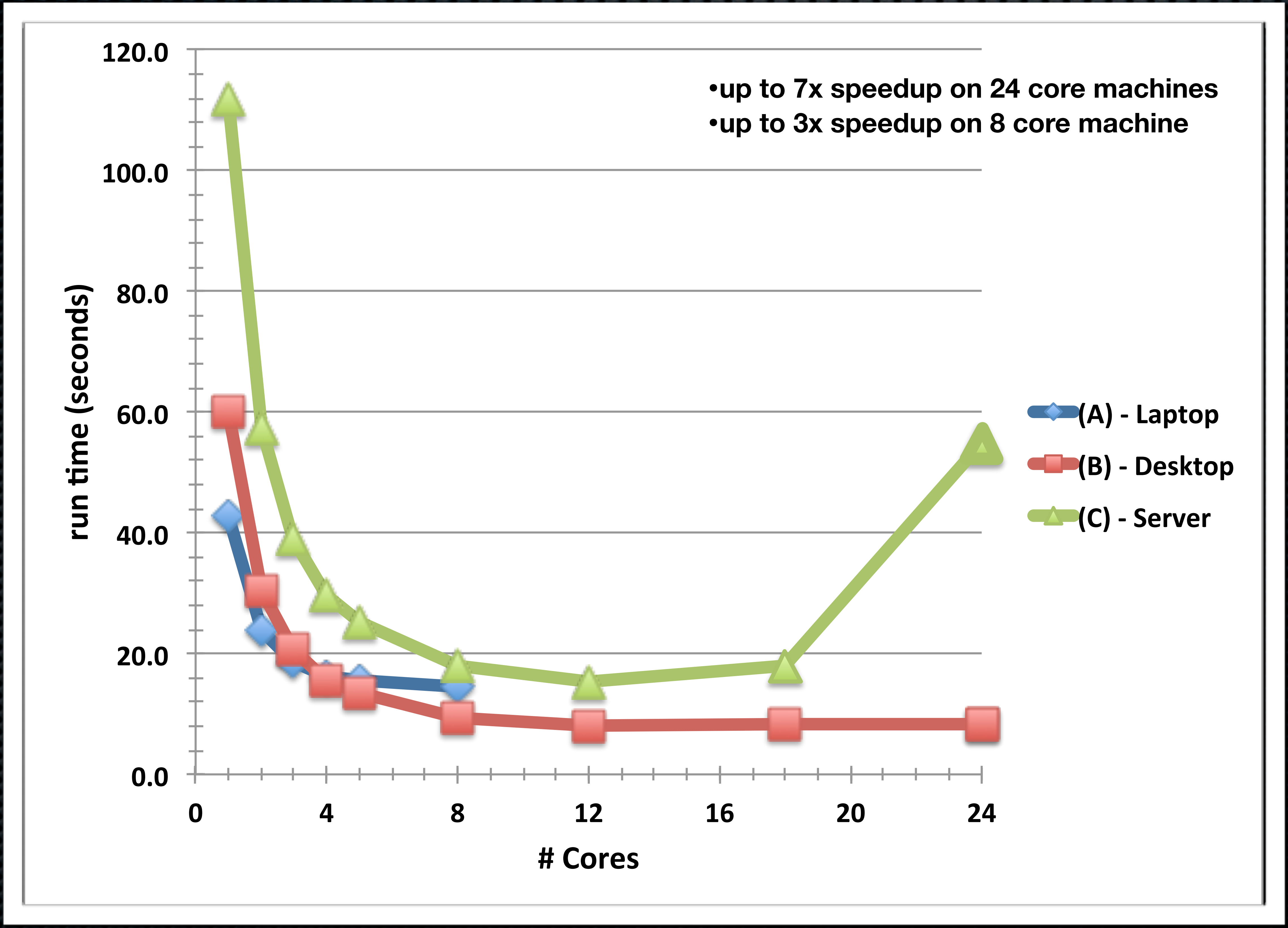


start
g=1…G^{max}
Start – Generation g
i=1…N
Create trial population member i
Compute objective values for trial member
Decide to keep: original, trial, or both
PARALLELIZED
Reduce population size to N
end

Experimental Hardware

System	Description	Processor	RAM	cores	
A	Laptop – MacBook Pro (2012 retina display)	2.7 GHz Core i7	16 GB	8	
B	Desktop – Mac Pro (2011)	2x 2.93 GHz Xeon X5670	64 GB	24	
C	Linux server Sunfire x4450 (2009)	4x 2.66 GHz Xeon X7460	128 GB	24	

Results —



•up to 7x speedup on 24 core machines
•up to 3x speedup on 8 core machine
120.0
100.0
80.0
60.0
40.0
20.0
0.0
run time (seconds)
0
4
8
12
16
20
24
Cores
(A) - Laptop
(B) - Desktop
(C) - Server

Results

- Best speedup is substantial:
  - 3x on 8-core machine
  - 7x on 24-core machine
- Using more than 1/2 the reported # cores is not beneficial
- Why is the Linux server proportionately worse when > 12 cores are used? (using 24 cores is no better than 2, and much worse than 12)
  - Memory bandwidth limitations has been reported as limiting factor in other similar work
  - particularly problematic in older server with slower RAM

Conclusions

- Parallelizing for multi-core hardware via Java 7 library features
  - easy to implement
  - can provide a major performance boost
  - some suggestions included in paper
- We are planning to configure as the default computational mode for the DSN long-range planning engine
- Next stages of LAPS development are less on performance than solution quality and visualization

Thank you!