6.189 Multicore Programming Primer, January (IAP) 2007

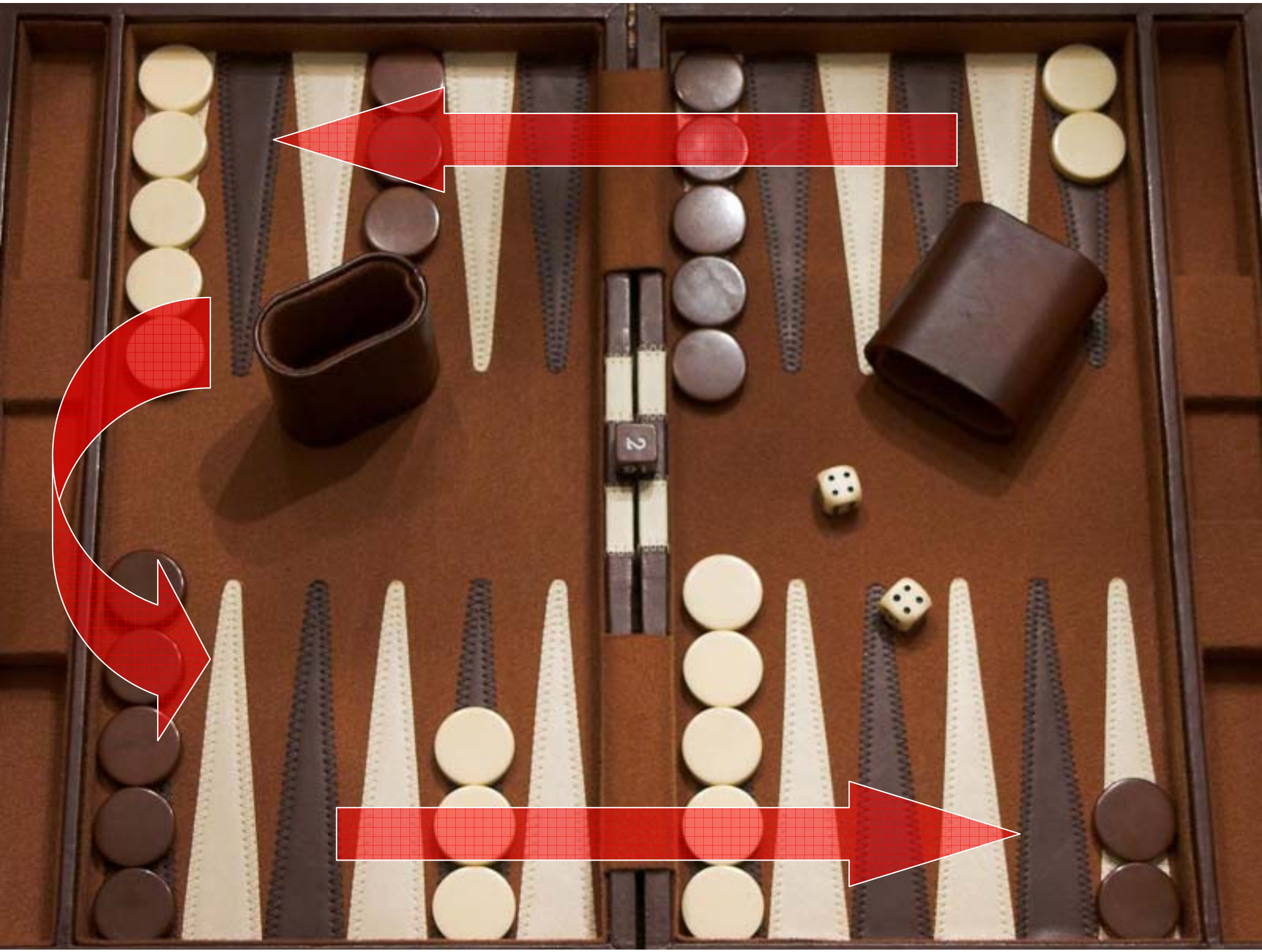
6.189 IAP 2007

Student Project Presentation

Backgammon Tutor

Backgammon Tutor

Eddie Scholtz and Mike Fitzgerald
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
February 1, 2007


2

Goals

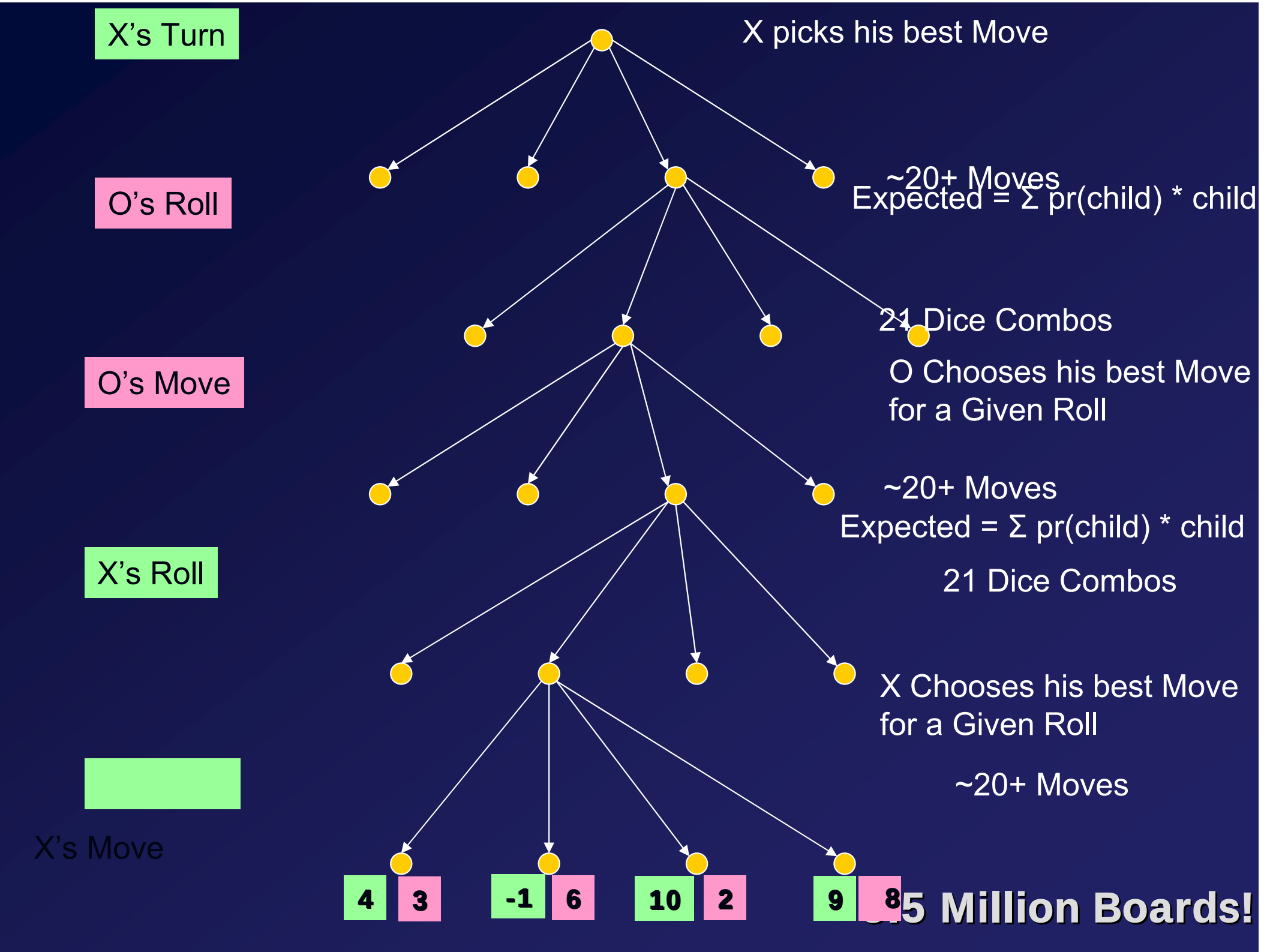
- Implement the rules of backgammon
- Create/Find a function that evaluates how “good” a board is
- Parallelize the evaluation of future board states in order to determine help determine what the best move is this turn
- Teach the player by suggesting a better move and explaining why it is a better move

Board Evaluation

- Static Evaluator
  - Linear sum of weighted board features
  - 1979 – BKG 9.8 (Hans Berliner – CMU) beat ruling world champion
    - Adjusted weights as game progressed
- Neural Nets
  - Traditionally most successful
  - Trained over 100,000+ games
  - Pubeval – Gerry Tesauro, IBM Research 1993
    - Released as benchmark evaluator that produces play at the intermediate level

Search

- Look at future turns in order to choose best move now
- Large Branching Factor
  - Checkers 10
  - Chess 35-40
  - Backgammon 400
- Uncertainty of future dice rolls (21 possible combinations)
- Pubeval is not zero-sum
- Does searching deeper produce a better play?
  - Most papers say search is less important than a good evaluator
  - Search produces slightly better play – Can still make a big difference

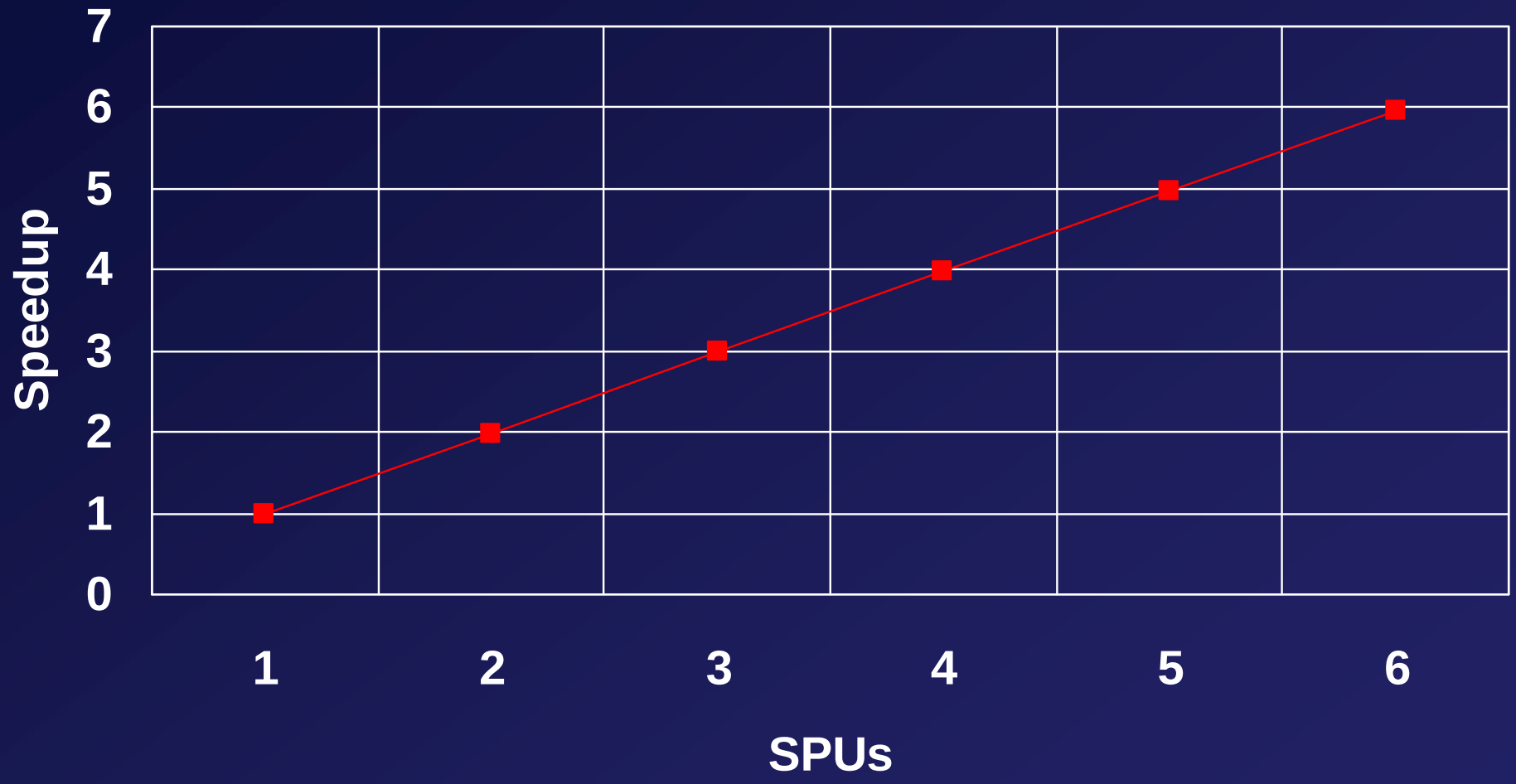

X's Turn
X picks his best Move
O's Roll
~20+ Moves
Expected = Σ pr(child) * child
21 Dice Combos
O's Move
O Chooses his best Move
for a Given Roll
~20+ Moves
Expected = Σ pr(child) * child
X's Roll
21 Dice Combos
X Chooses his best Move
for a Given Roll
~20+ Moves
X's Move
4
3
-1
6
10
2
9
8
5 Million Boards!

Parallelizing Board Evaluation

- Millions of leaf nodes that each represent a board state
- Attempt to split evenly between SPUs
- Each has a multiple of 4 boards
- All boards (~170,000 in benchmarks) can't go over to the SPU at once
- SPU knows how much it has to process, takes as much as it can, evaluates, returns, and gets more
- Each should finish at roughly the same time
- SIMDize code to evaluate 4 boards at once
- Double buffer so we can DMA and compute at the same time

Performance – Evaluating 1 Million Boards

Speedup – Evaluating 1 Million Boards



Demo

Development Techniques

- Get sequential code working correctly on single core
  - Squash bugs and memory leaks
- Implement parallel code as sequential code to make sure algorithm works
- Convert parallel code to run on 1 SPU
- Get code working on all 6 SPUs
- Most debugging done with printf statements

Challenges

- We aren’t backgammon experts or even intermediate players
- Getting the game to a playable state took a good chunk of our total time
- Managing search tree
- Memory management
- Cell Debugging
- Bit packing the boards so when we are storing millions of them they fit into memory

Future Work

- Finish move tree traversal after board evaluation
- Move tree pruning (beam search)
- Parallelize move tree creation and traversal
- SIMDize and buffer board evaluation
- Training Board Evaluators
- Monte Carlo approach to finding the best move
- Explaining in English why one move is better than another