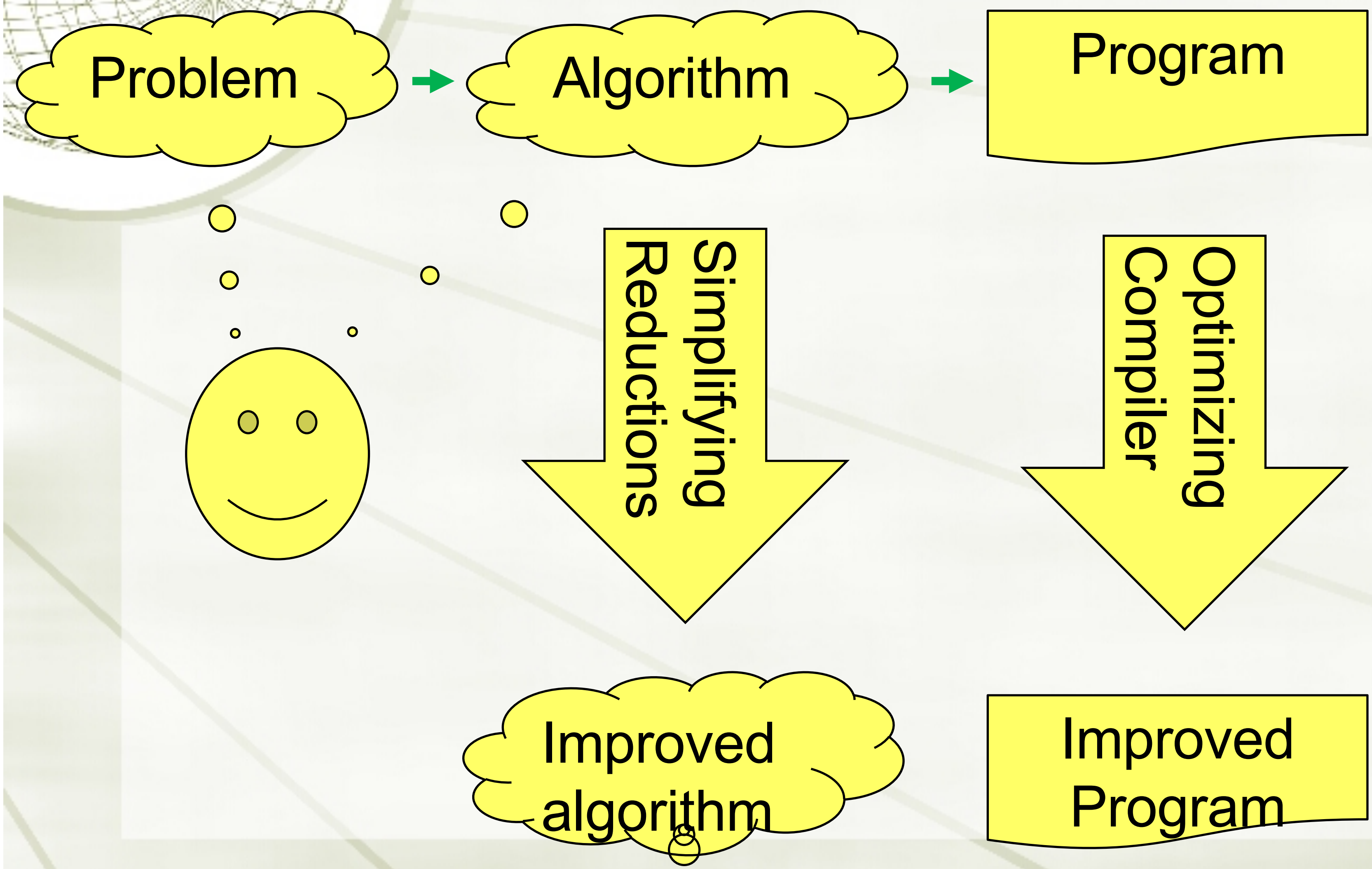
Simplifying Dependent Reductions

Sanjay Rajopadhye

Colorado State University

Overview

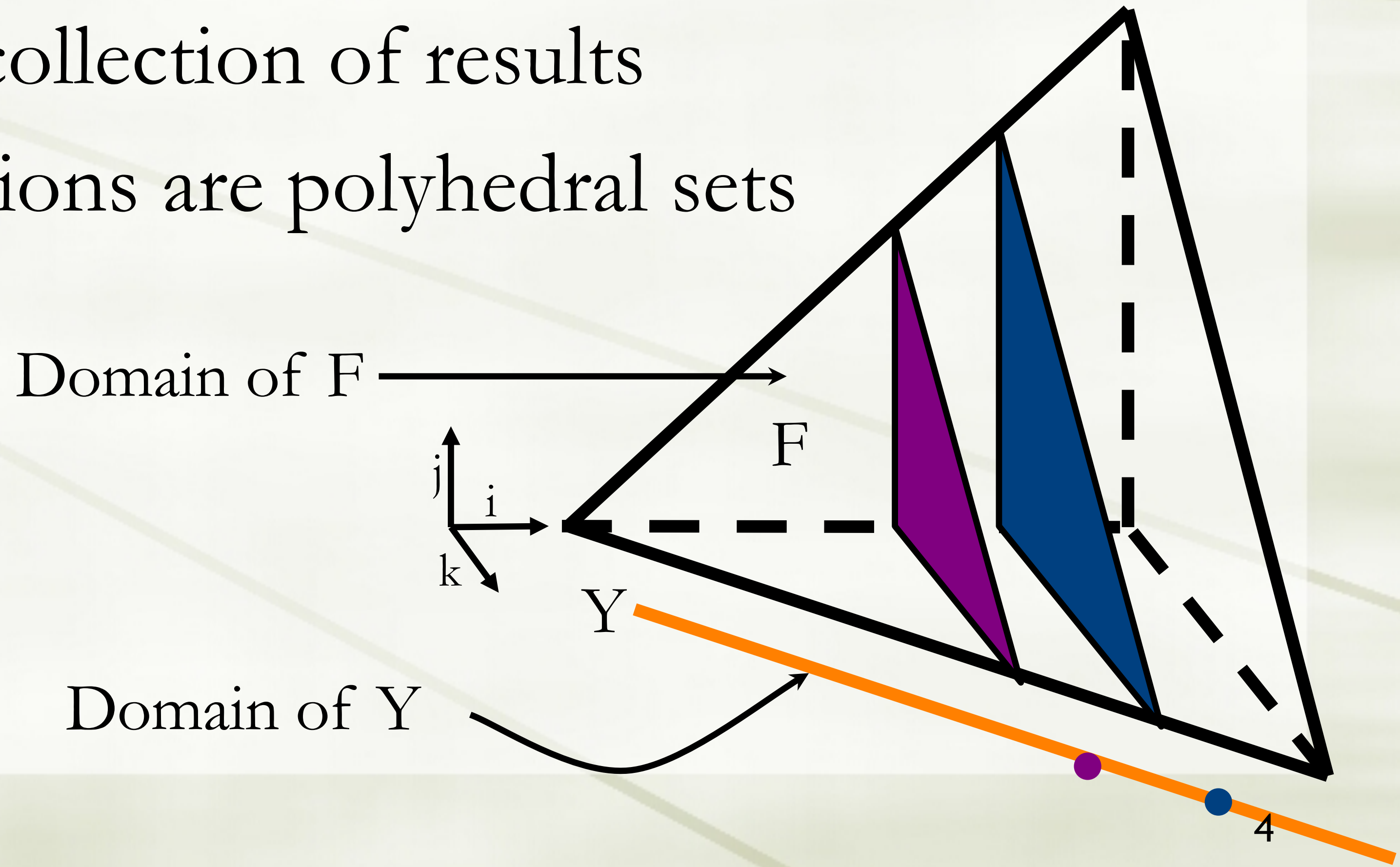

Problem
Algorithm
Program
Simplifying Reductions
Optimizing Compiler
Improved algorithm
Improved Program

Reductions

- A reduction is an associative and commutative operator applied to collections of values to produce a single or collections of results

Reductions

- A reduction is an associative and commutative operator applied to collections of values to produce a collection of results
- Our collections are polyhedral sets



Simple Example (scan)

- Compute an array Y given by the equation

$$Y_i = \sum_{k=0}^{i} X_j$$

```
for i = 0 to n {
    Y[i] = 0;
    for j = 0 to i
        Y[i] += X[j]
}
```

```
Y[0] = X[0];
for i = 1 to n {
  Y[i] = Y[i-1]+X[i]
```

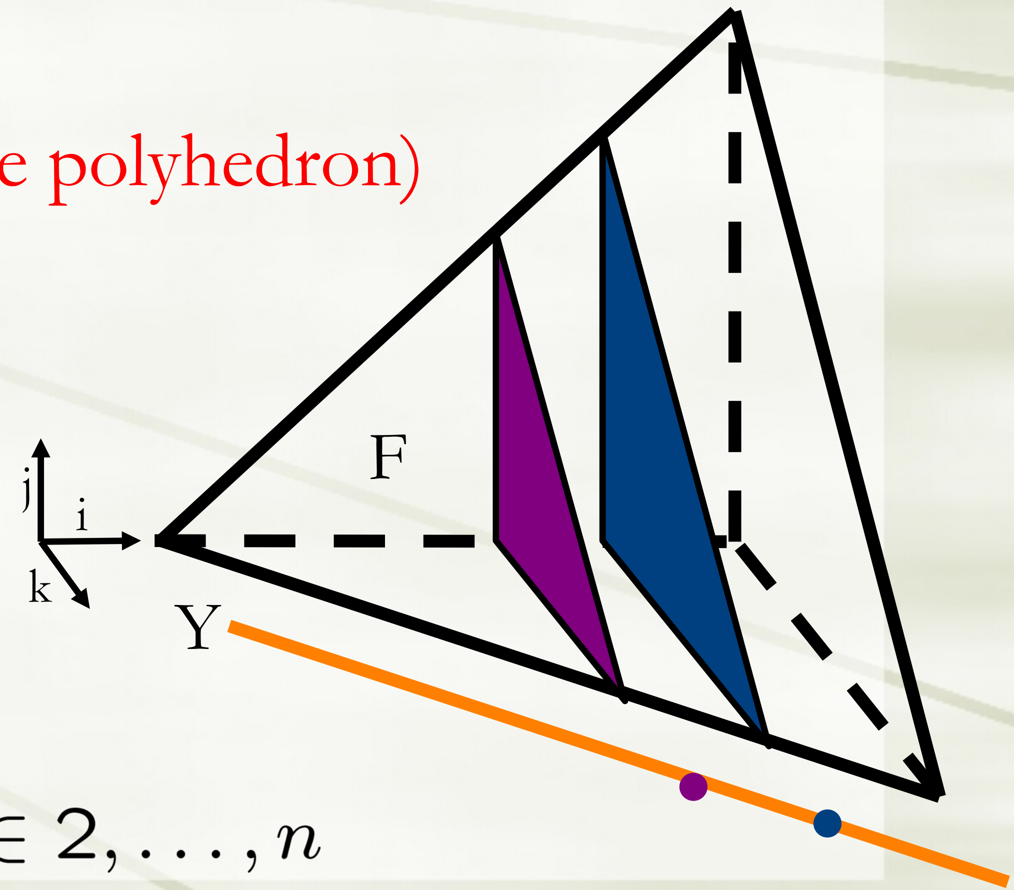
Outline

- Introduction and Problem Definition
- Sharing
- Simplification
  - Multidimensional Simplification
- Gautam Rajopadhye algorithm
- Dependent Reductions, what's the problem?
- Coupling Scheduling and Simplification
- Related Work & Conclusions

Representation

- Three equivalent forms of representation
  - Geometric
  - Loops (bounds define the polyhedron)

```
for i = 1 to n {
 Y[i] = 0;
 for j = 1 to i-1
  for k = 1 to i-j
   Y[i] += F[i,j,k]; }
```

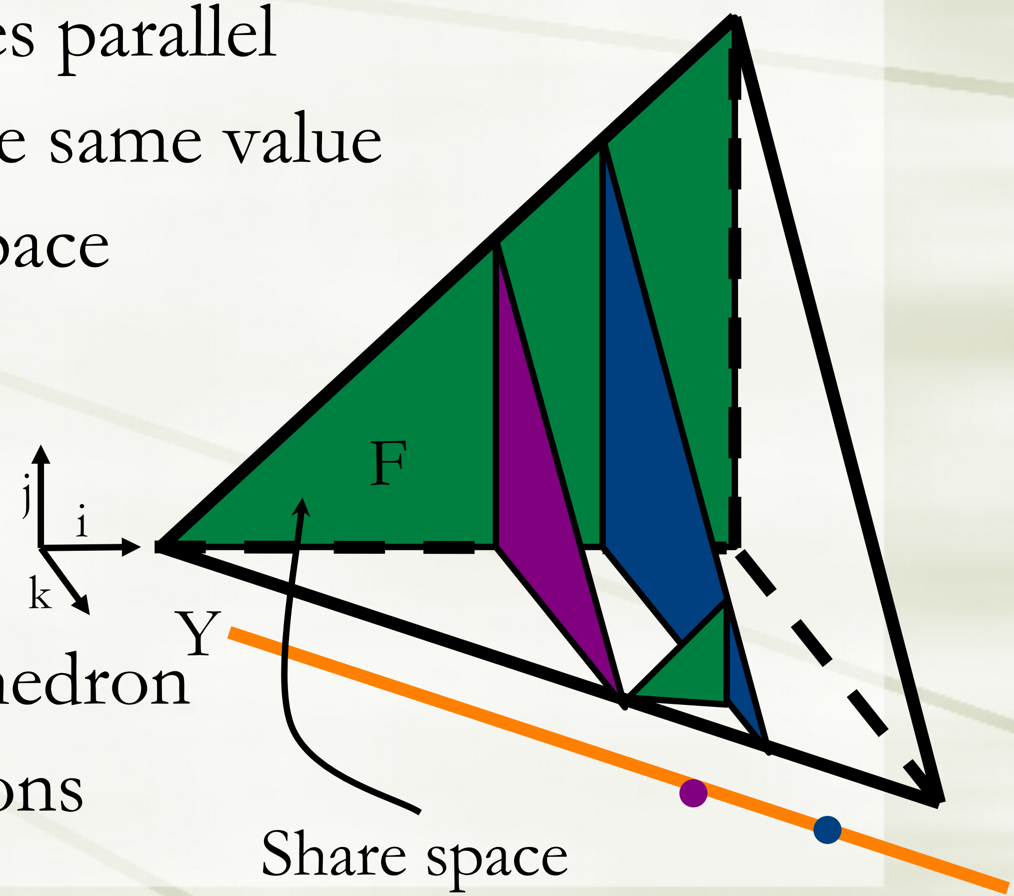


  - Equations

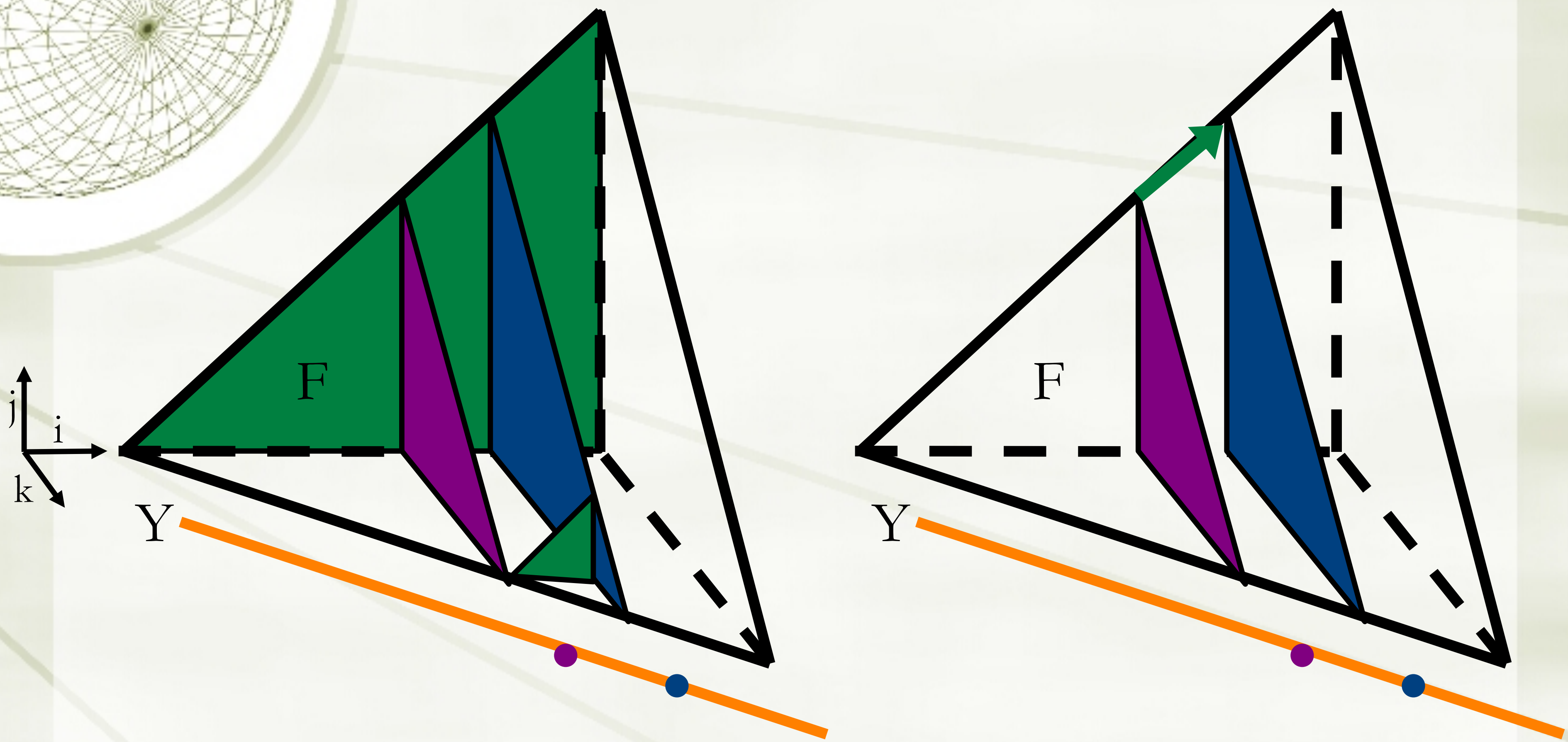
$$Y_i = \sum_{j=1}^{i-1} \sum_{k=1}^{i-j} F_{i,j,k}, \; i \in 2, \ldots, n$$

Sharing

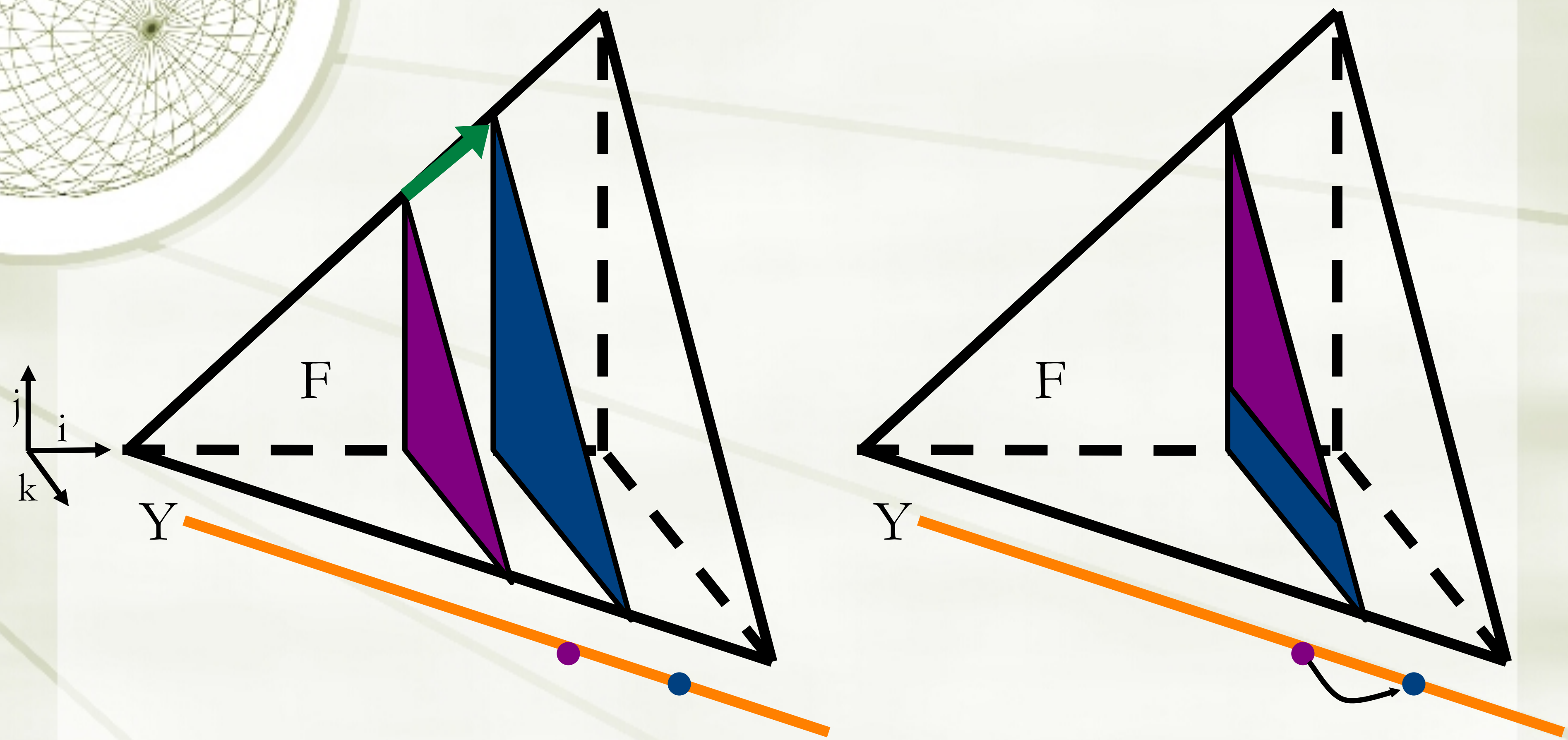
- If $F_{i,j,k} = X_k$
  - All index points on planes parallel to the {i,j} plane have the same value
  - {i,j} is called the share space
    - Denoted by green
  - Aim to replace this polyhedron by one of lesser dimensions



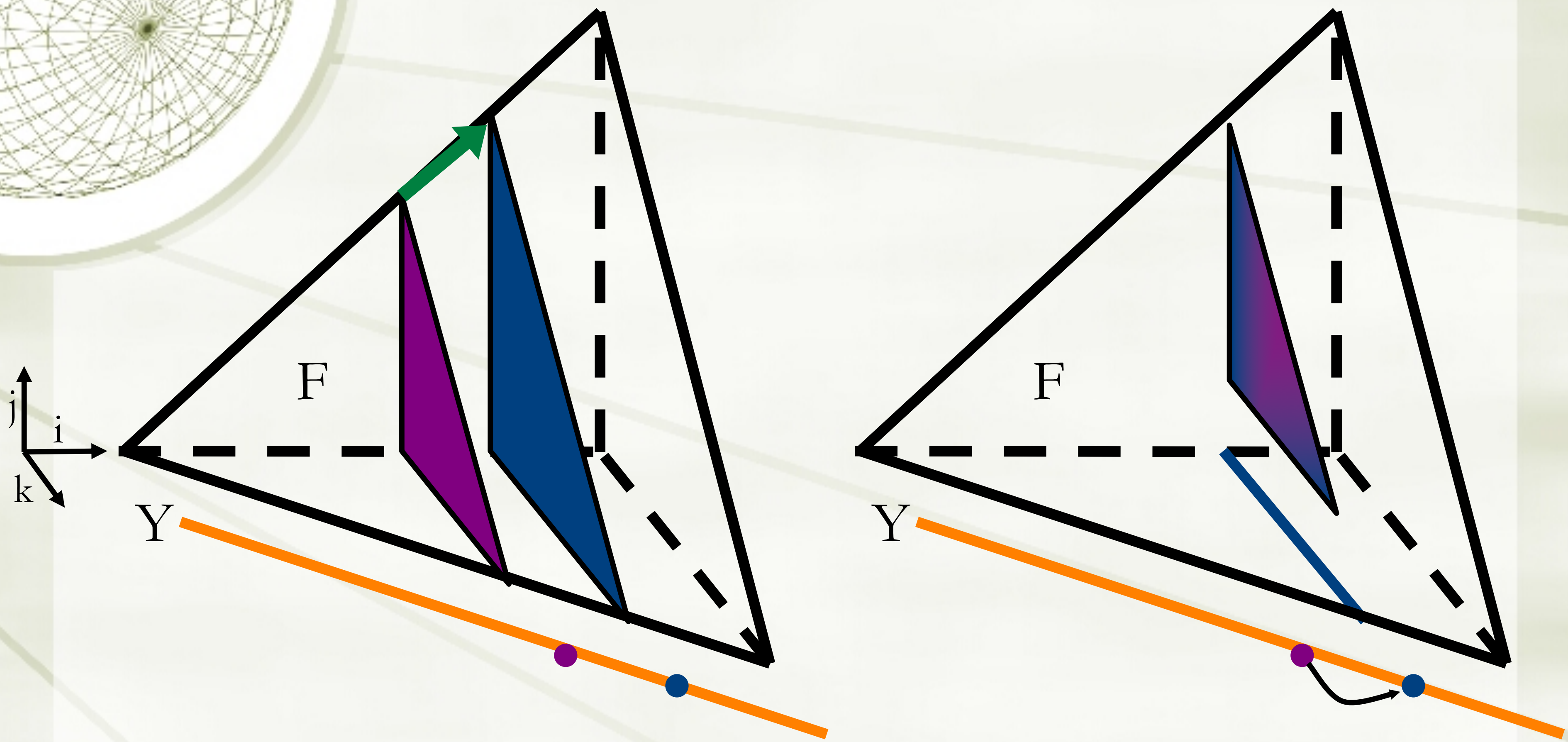
Simplification



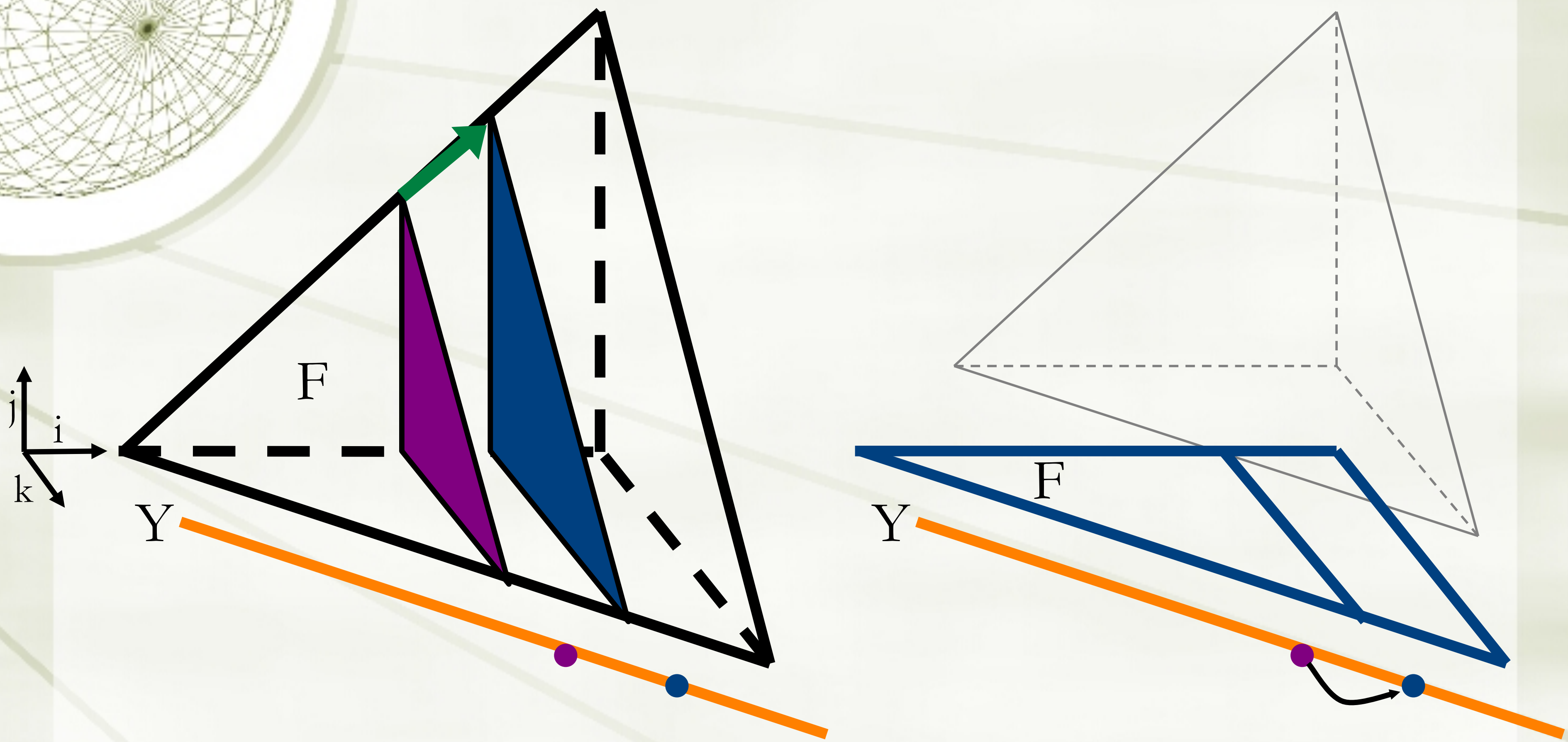
Simplification



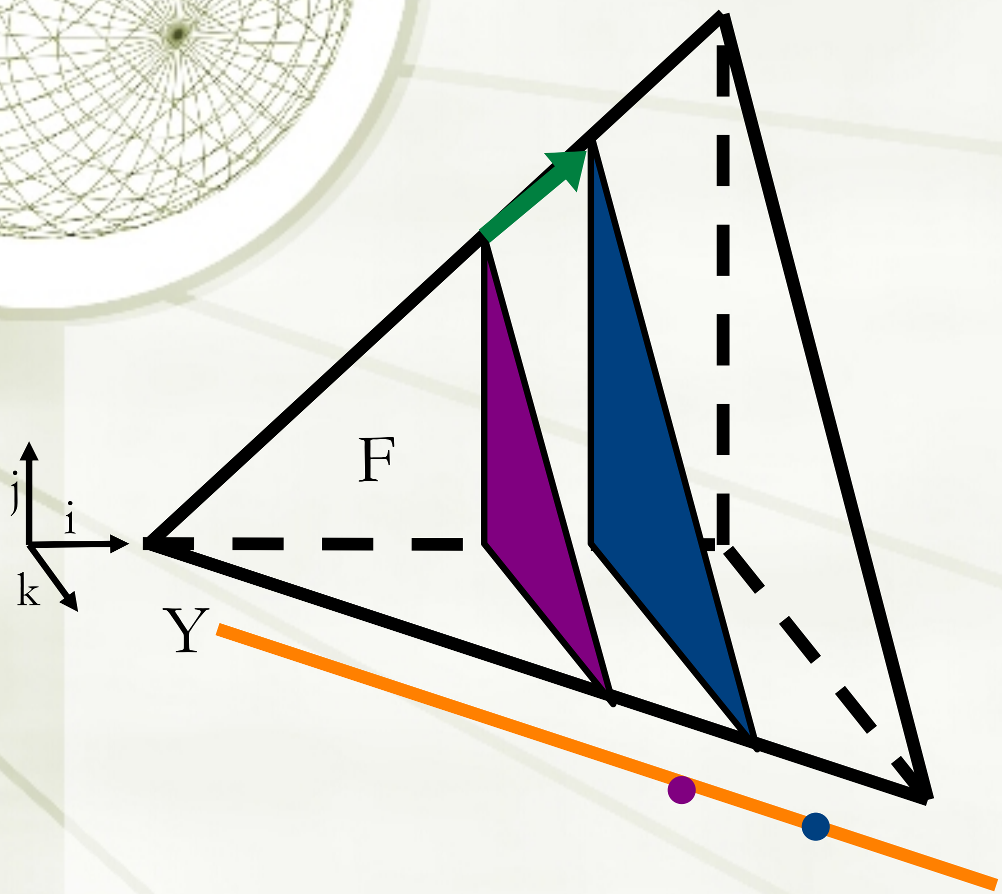
Simplification

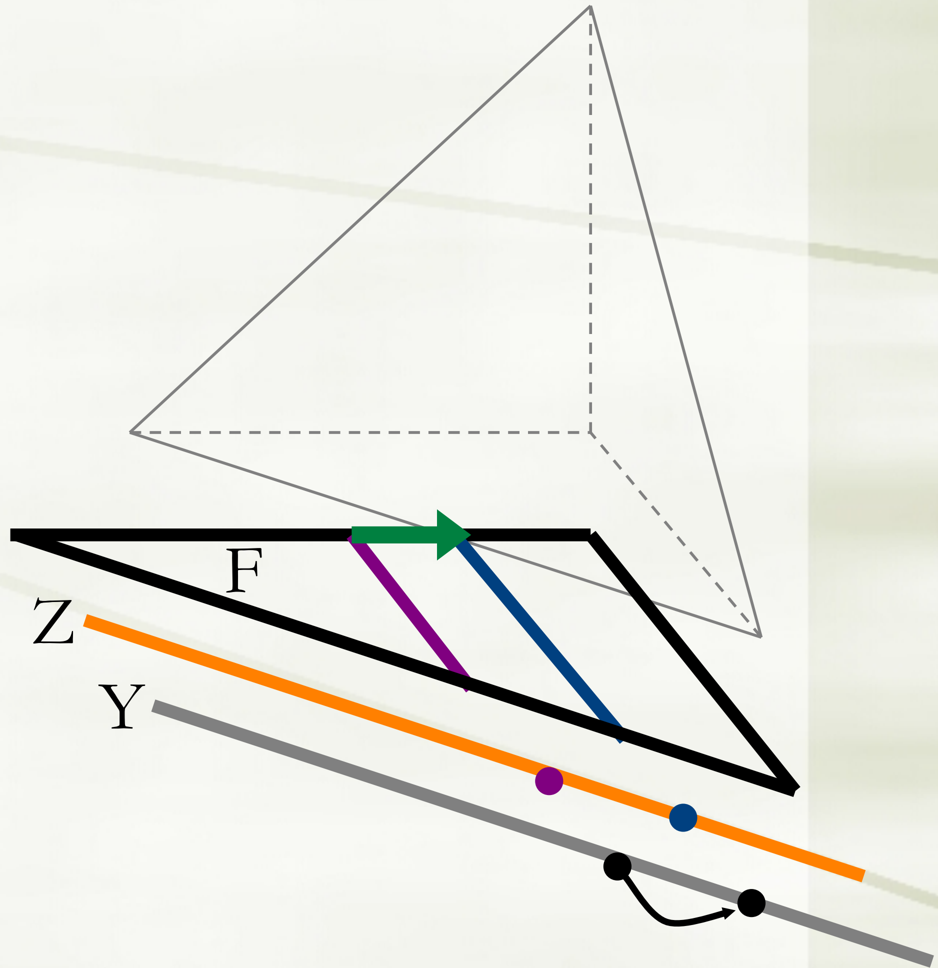


Simplification

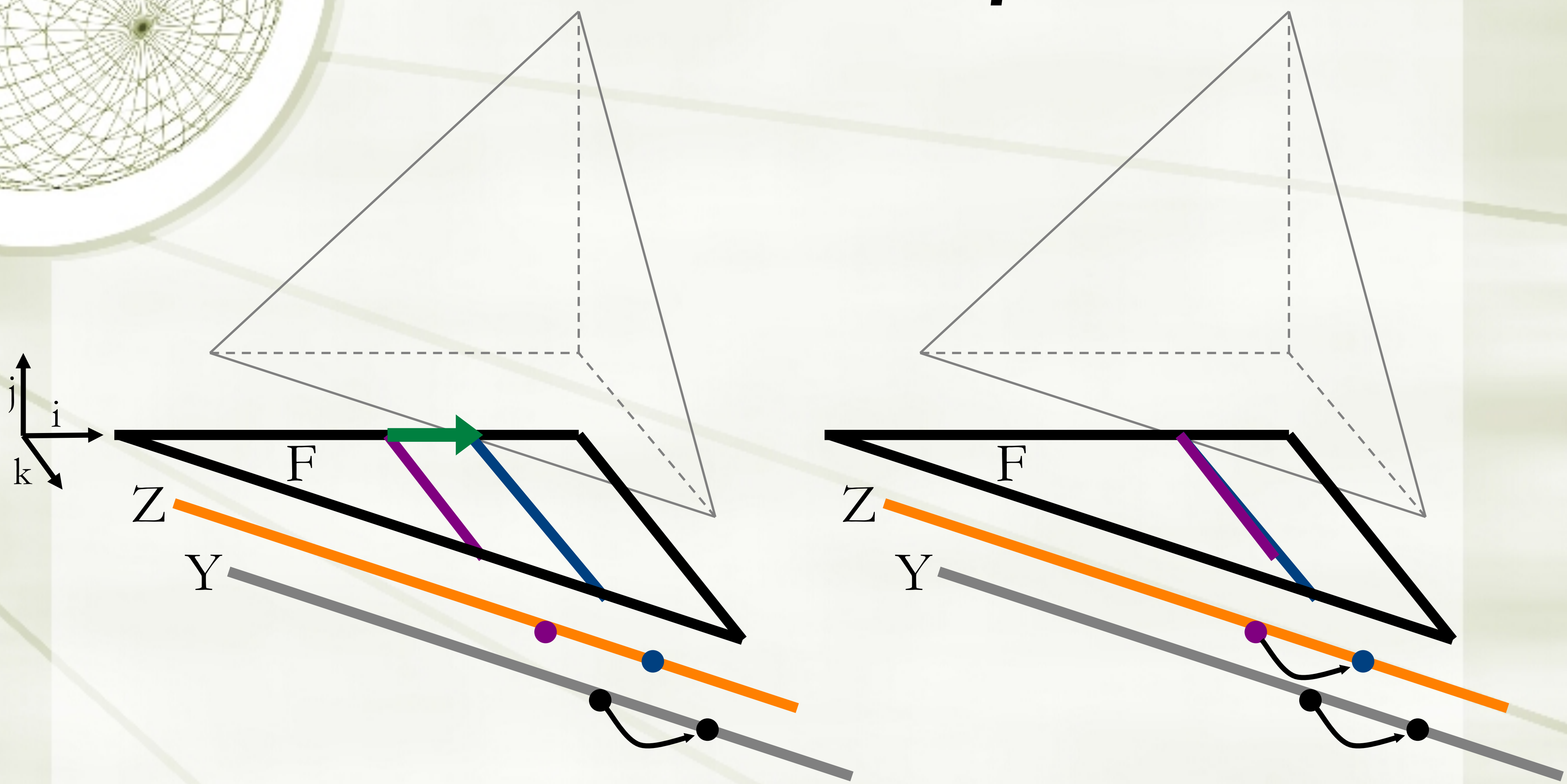


Multidimensional Simplification

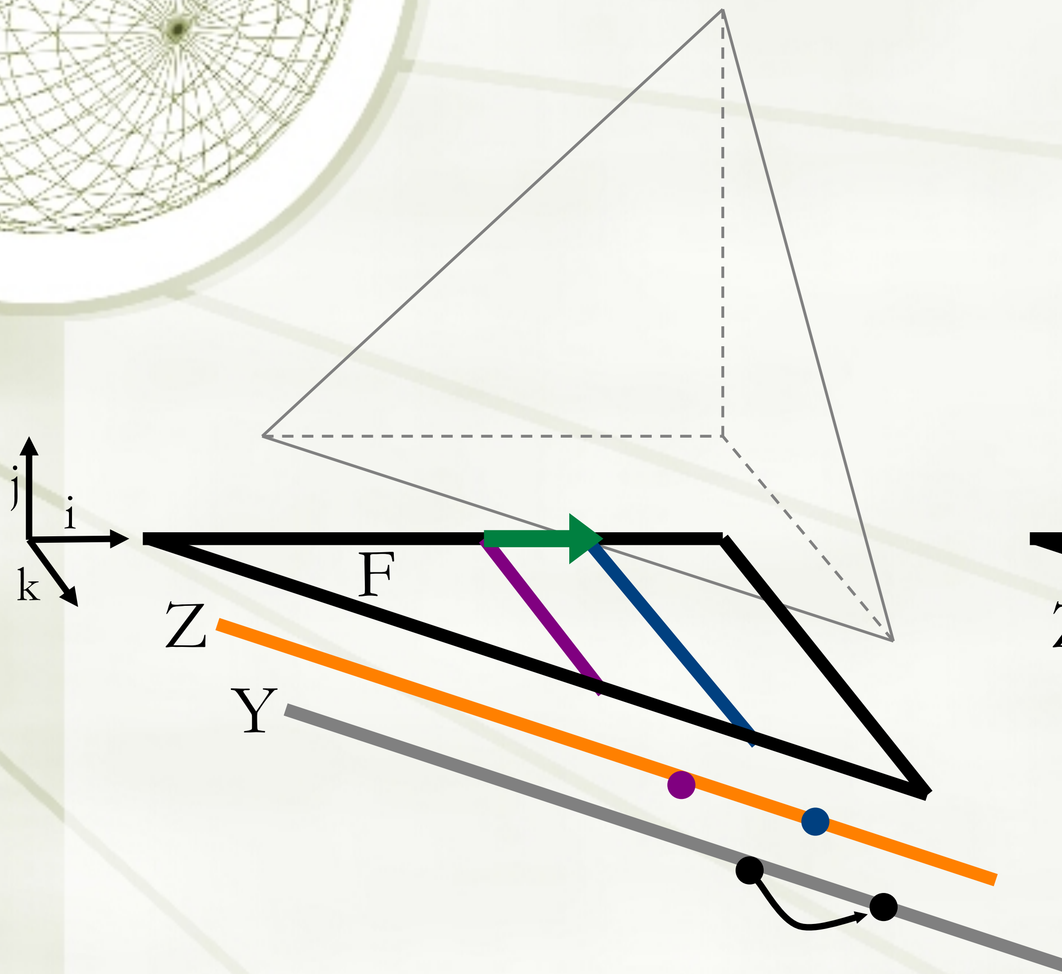


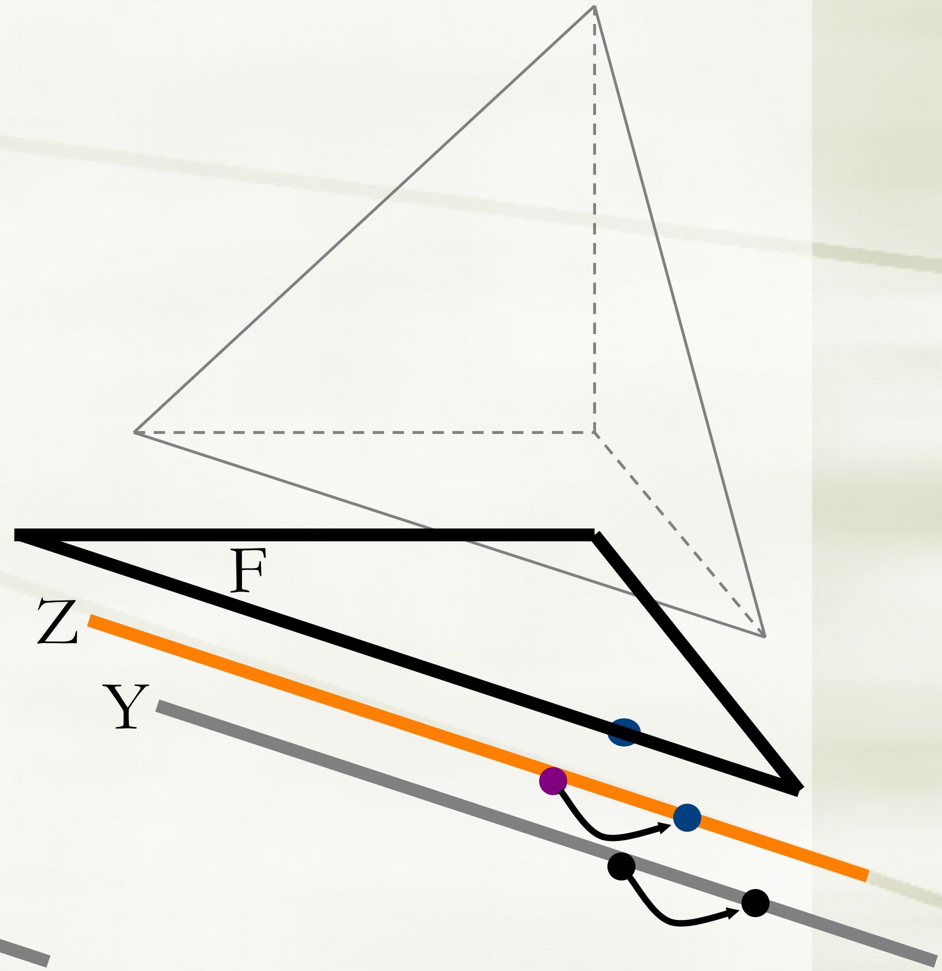


Multidimensional Simplification

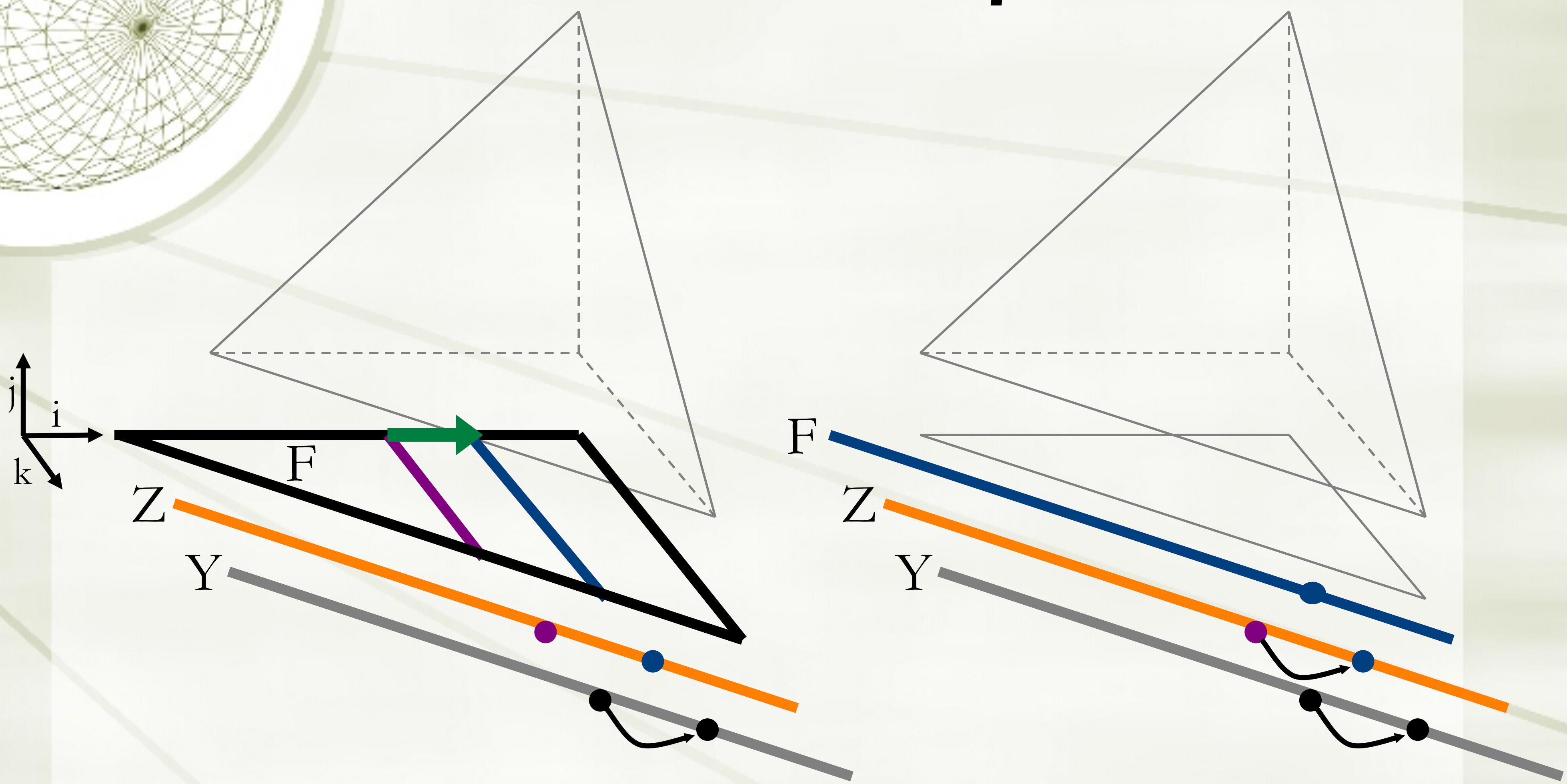


Multidimensional Simplification

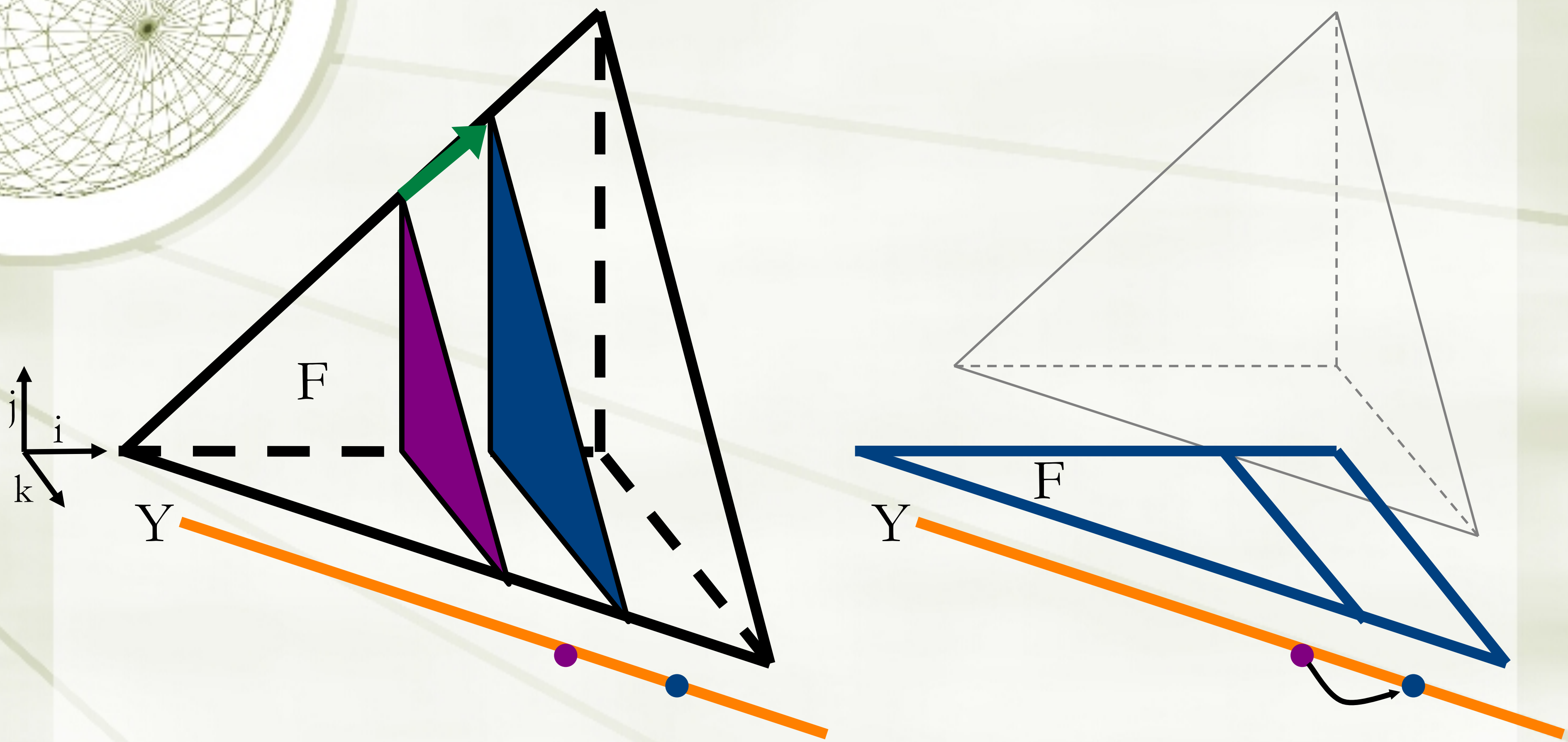




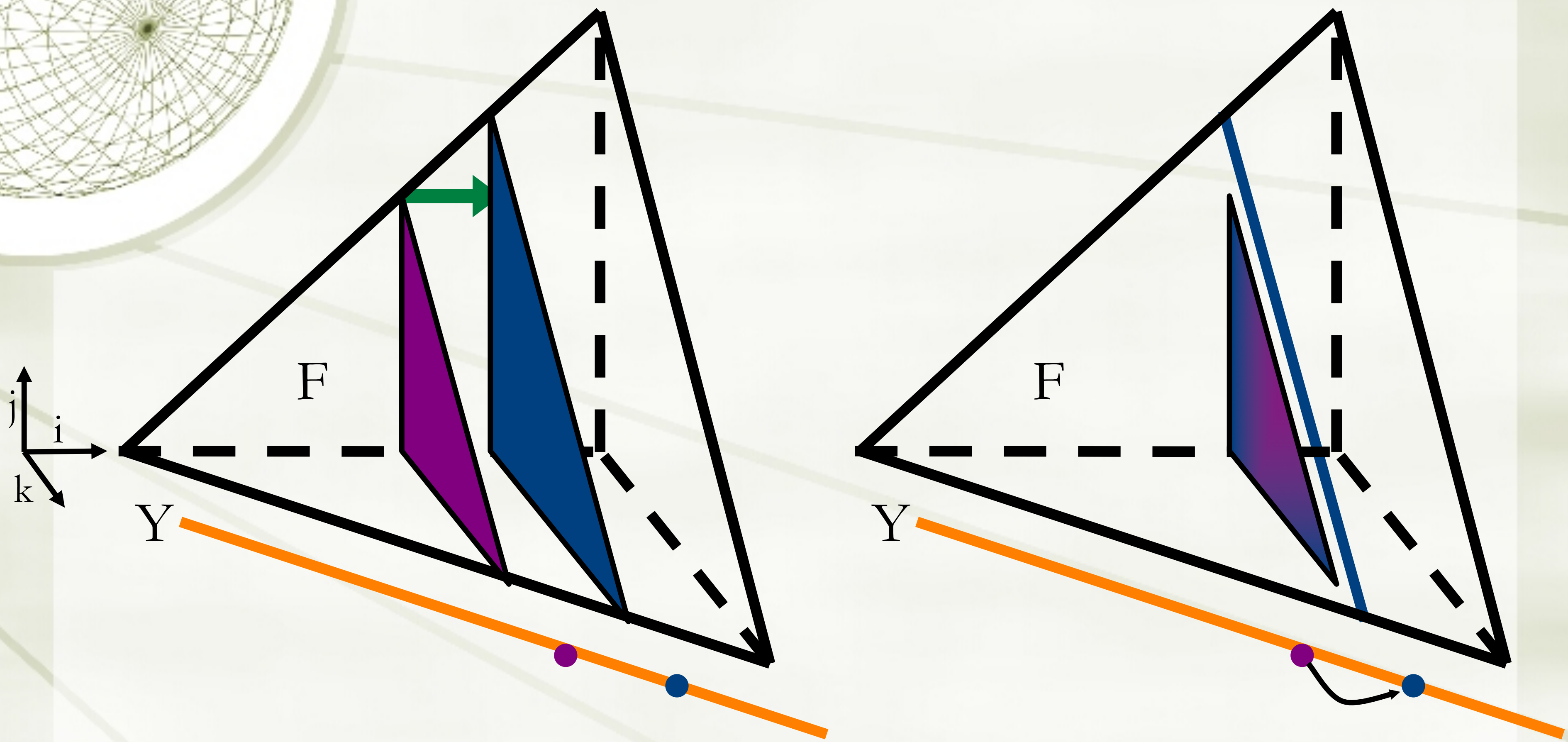
Multidimensional Simplification



Infinite Space



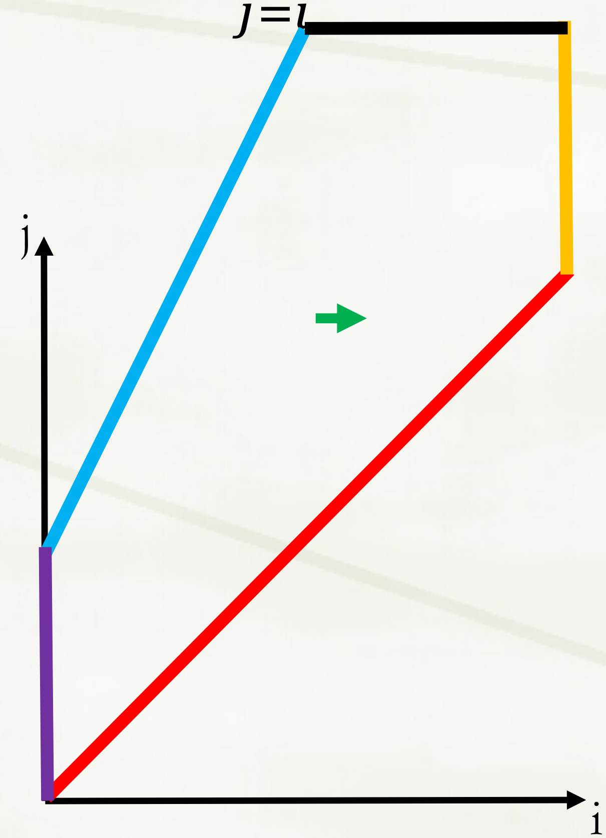
Another Option



GR 2006 Algorithm

$$Y_i = \sum_{j=i}^{\min(2i+N-1,3N-1)} X_j$$

- Preprocessing:
  - Determine the share/reuse space
  - Construct the face lattice
- Pick a reuse vector ρ
- Translate domain along ρ
- Delete the intersection, retain residual computation on the differences (facets)
- Label each facet as:
  - Boundary
  - Inward/outward/invariant (function of ρ)
- Ignore outward boundary & invariant facets
- Accumulate inward boundary (initialize)
- Add inward facets
- Subtract outward facets
- Recurse on each facet



GR 2006 Algorithm

Infinitely many choices

- Only finitely many labels
- All choices of ρ that yield the same facet labels are equivalent for complexity reduction
- Only finitely many choices at each level
  - May need to backtrack
- All roads lead to Rome: if reduction operator admits an inverse,
  - All available dimensions can be fully exploited
  - All choices of reuse vectors to exploit are equivalent

Dependent Reductions

- What if X depends on Y?

$$Y_i = \sum_{j=i}^{\min(2i+N-1,3N-1)} X_j$$

$$X_i = f(Y_{j-1})$$

- Not all reuse vectors are legal
  - Cyclic dependences
- Couple simplification with scheduling
  - Polyhedral scheduling is well known when dependences are given
  - But reuse vectors are unknown (chosen as the algorithm recurses down the face lattice)

Solution

- Key insight: The feasible space of legal schedules is a
  finitely generated (w generators $\theta_1 \dots \theta_m$)
  blunt (i.e., does not contain the origin)

cone

- Start with the feasible space of all schedules of the original program
- When choosing the reuse vector ρ at each face, make sure that the cone does not become empty
  - At least one of the generators satisfies that $\theta_i \, \rho$ is non-negative
  - Leads to m disjunctions, but only finitely many choices
  - Retains optimality of GR 2006

Related Work

- Roychowdhury 1988
- Delosme Ipsen 1985
- Yang Atkinson and Carbin [POPL 2021]
  - First to formulate the problem
  - Many practical use cases from probabilistic programming
  - Formulated solution as bilinear programming plus simple heuristic that works in practice.

Conclusions

- Simplifying reductions has practical benefits
- Dependences add a new twist (whole program analysis, not just one equation)
- We can have optimal simplification even with dependences