


MILTON BABBITT'S COMPOSITIONAL PROCESS: A COMPUTATIONAL MODEL

Brian M. Bemman

MT Colloquium

Milton Babbitt (1916–2011)

- Avant-garde composer of twelve-tone, atonal and serial music – “beyond” tonal repertoire of Bach, Mozart, etc. 
- Sought mathematical methods of composition – continuation of practices developed by Schoenberg
- Third period of compositional style notable for the creation of the **all-partition array** structure

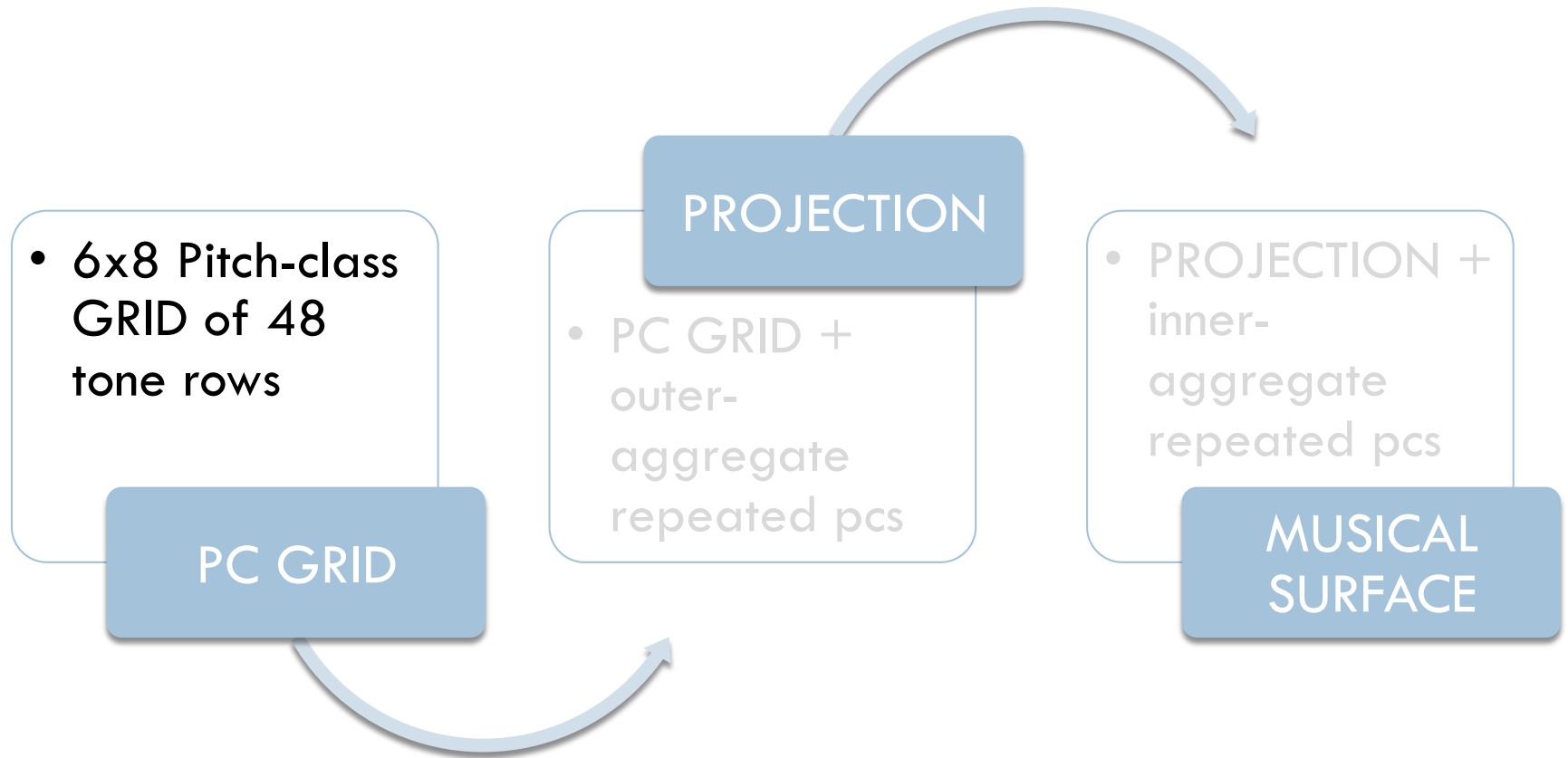
Purpose

- Musical analysis – description of the structure of a musical piece
- Most efforts (by human or machine) take as input the musical surface (i.e. score) and produce some analysis
- This research represents efforts to do the reverse – take as input an encoding of an analysis (i.e. the all-partition array) and produce the musical surface

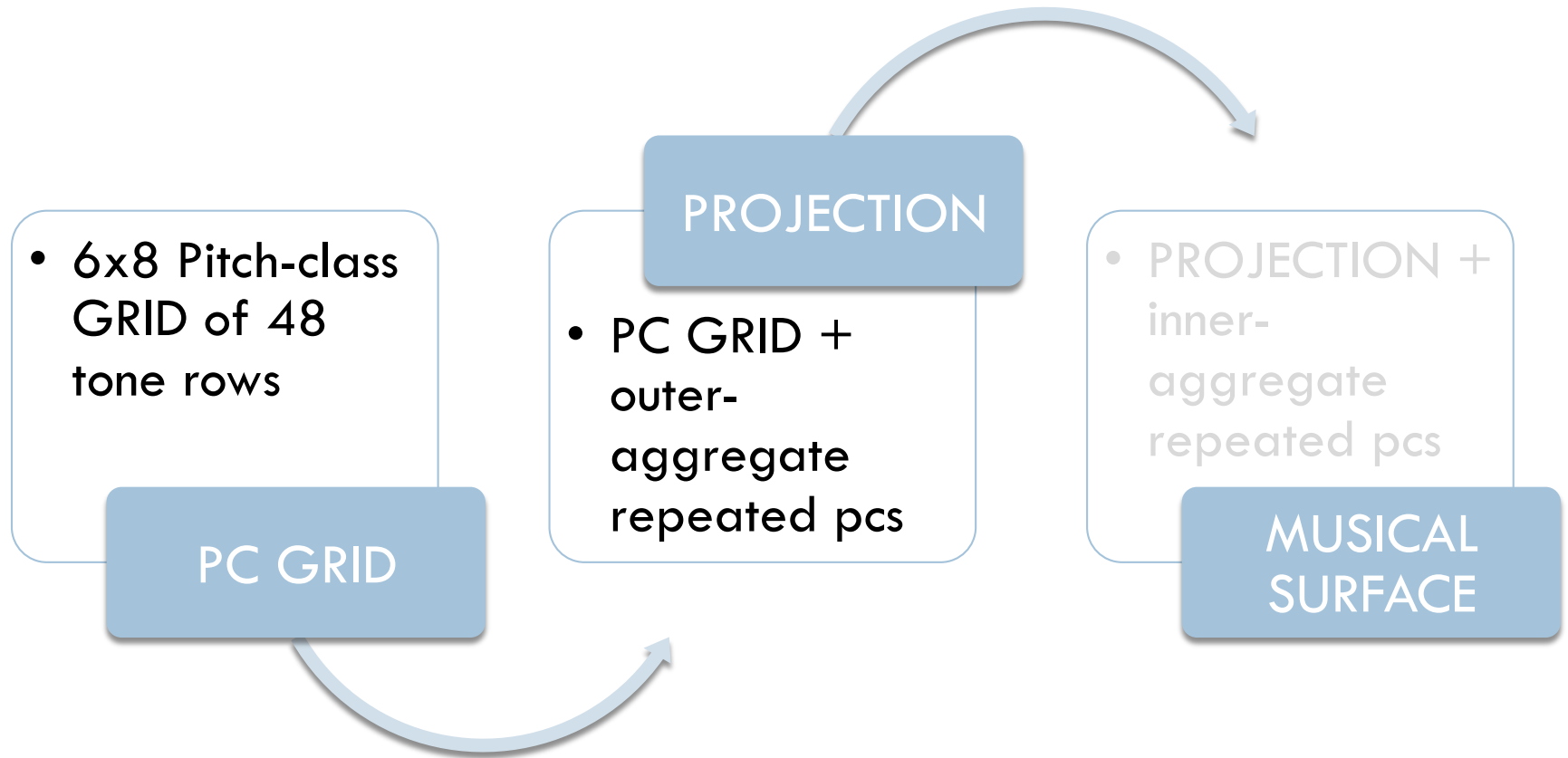
Basic Terminology

- **Pitch class** = set of all musical pitches a whole number of octaves apart – C_2, C_3, C_4, C_5 etc...
- Pitch classes can be represented as integers modulo 12 – $C = 0, C\# = 1, D = 2$ etc...
- An **aggregate** is a collection of all twelve pitch classes.
- A **tone row** is some ordered set of the aggregate – $\langle 0, 1, 1, 6, 5, 4, 8, 1, 9, 2, 10, 7, 3 \rangle$

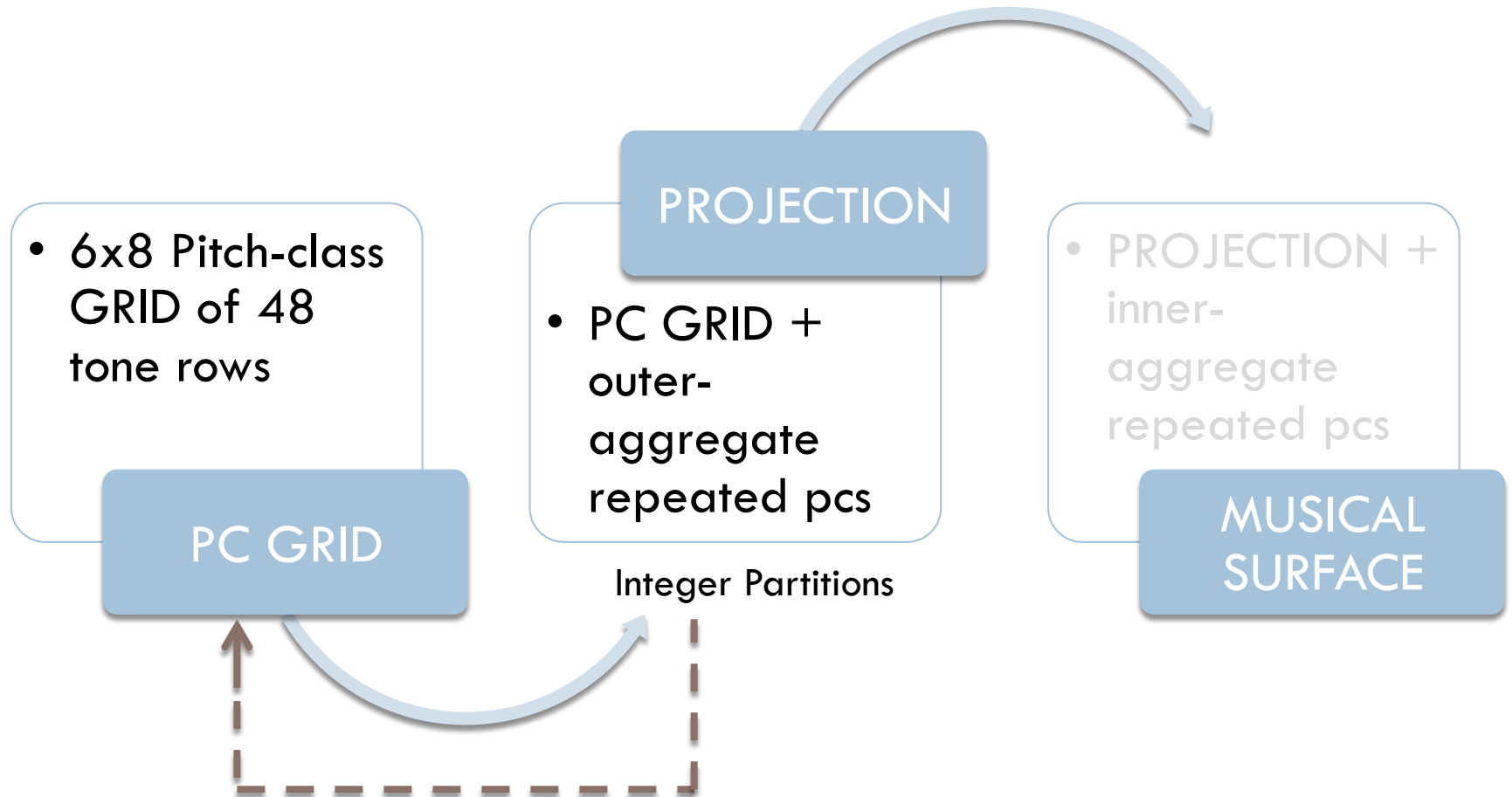
Compositional Process



Compositional Process



Compositional Process



Compositional Process

All-partition array

- 6x8 Pitch-class GRID of 48 tone rows

PC GRID

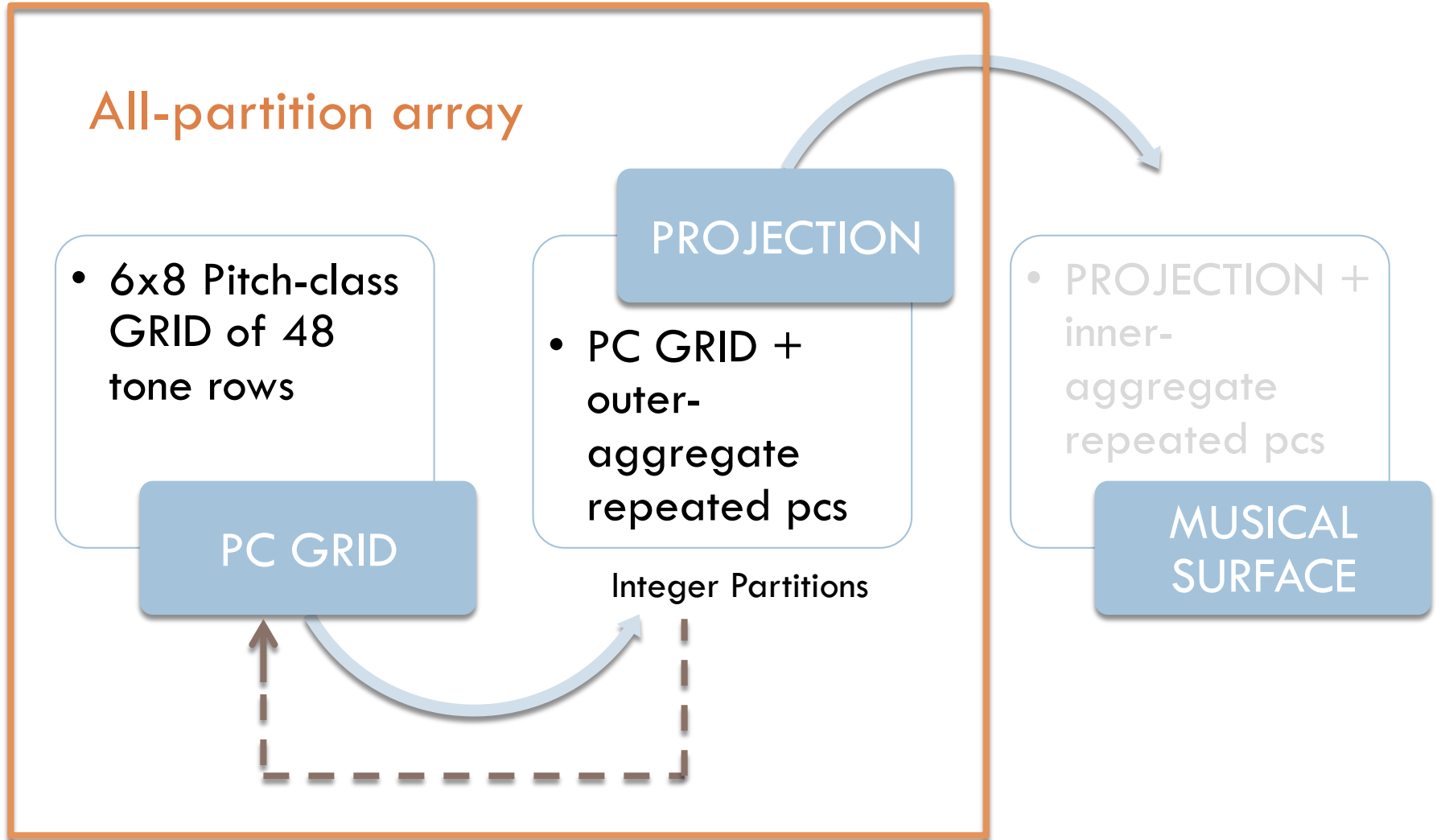
- PC GRID + outer-aggregate repeated pcs

Integer Partitions

PROJECTION

- PROJECTION + inner-aggregate repeated pcs

MUSICAL SURFACE



PC GRID

6x8 grid of 48 tone rows = 576 pcs

R _{1,1}	R _{1,2}	R _{1,3}	R _{1,4}	R _{1,5}	R _{1,6}	R _{1,7}	R _{1,8}
R _{2,1}	R _{2,2}	R _{2,3}	R _{2,4}	R _{2,5}	R _{2,6}	R _{2,7}	R _{2,8}
R _{3,1}	R _{3,2}	R _{3,3}	R _{3,4}	R _{3,5}	R _{3,6}	R _{3,7}	R _{3,8}
R _{4,1}	R _{4,2}	R _{4,3}	R _{4,4}	R _{4,5}	R _{4,6}	R _{4,7}	R _{4,8}
R _{5,1}	R _{5,2}	R _{5,3}	R _{5,4}	R _{5,5}	R _{5,6}	R _{5,7}	R _{5,8}
R _{6,1}	R _{6,2}	R _{6,3}	R _{6,4}	R _{6,5}	R _{6,6}	R _{6,7}	R _{6,8}

PC GRID

6x8 grid of 48 tone rows = 576 pcs

R _{1,1}	R _{1,2}	R _{1,3}	R _{1,4}	R _{1,5}	R _{1,6}	R _{1,7}	R _{1,8}
R _{2,1}	R _{2,2}	R _{2,3}	R _{2,4}	R _{2,5}	R _{2,6}	R _{2,7}	R _{2,8}
R _{3,1}	R _{3,2}	R _{3,3}	R _{3,4}	R _{3,5}	R _{3,6}	R _{3,7}	R _{3,8}
R _{4,1}	R _{4,2}	R _{4,3}	R _{4,4}	R _{4,5}	R _{4,6}	R _{4,7}	R _{4,8}
R _{5,1}	R _{5,2}	R _{5,3}	R _{5,4}	R _{5,5}	R _{5,6}	R _{5,7}	R _{5,8}
R _{6,1}	R _{6,2}	R _{6,3}	R _{6,4}	R _{6,5}	R _{6,6}	R _{6,7}	R _{6,8}

Column 1 of PC GRID

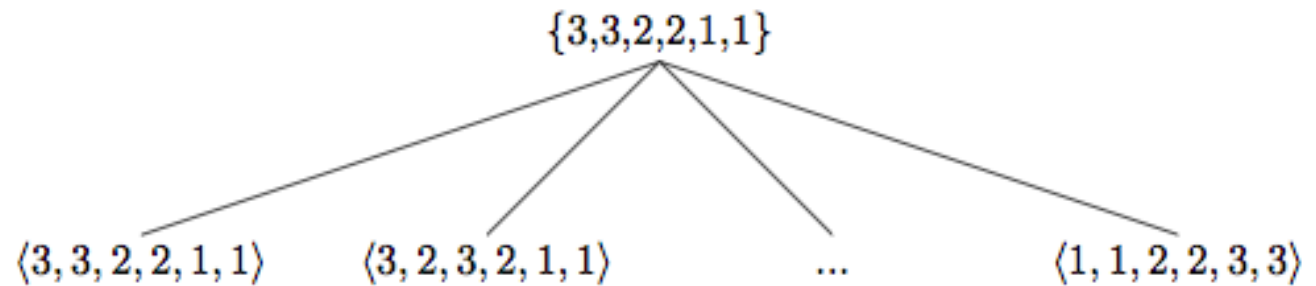
11	4	3	5	9	10	1	8	2	0	7	6
6	7	0	2	8	1	10	9	5	3	4	11
5	6	11	1	7	0	9	8	4	2	3	10
2	9	10	8	4	3	0	5	11	1	6	7
0	5	4	6	10	11	2	9	3	1	8	7
1	8	9	7	3	2	11	4	10	0	5	6

Integer Partition vs. Integer Composition

There are 58 such integer partitions.

There are 6,188 such integer compositions.

An integer partition is a representation of an integer n , as an *unordered* sum of k positive integers. If $n = 12$ and $k = 6$, one possible integer partition is



An integer composition is a representation of an integer n , as an *ordered* sum of k positive integers.


The role of integer partitions in the all-partition array

- Integer partitions are used to parse the PC GRID into a sequence of uniquely “shaped” *vertical* aggregates
- Must use all 58 integer partitions
- 58 partitions requires 696 pcs
- But... PC GRID contains only 576 pcs
- In order to go from PC GRID to PROJECTION we must insert 120 extra pcs...outer-aggregate repeated pcs

PC GRID to PROJECTION

Column 1 of PC GRID

Column 1 cross-section of PROJECTION

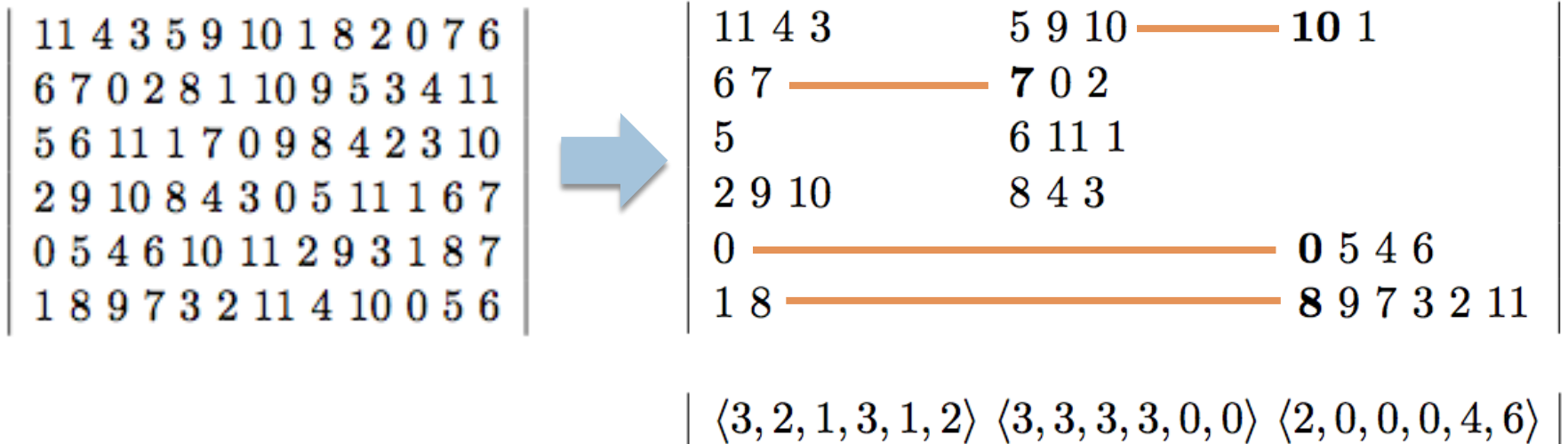
11 4 3 5 9 10 1 8 2 0 7 6		11 4 3	5 9 10	10 1
6 7 0 2 8 1 10 9 5 3 4 11		6 7	7 0 2	
5 6 11 1 7 0 9 8 4 2 3 10		5	6 11 1	
2 9 10 8 4 3 0 5 11 1 6 7		2 9 10	8 4 3	
0 5 4 6 10 11 2 9 3 1 8 7		0		0 5 4 6
1 8 9 7 3 2 11 4 10 0 5 6		1 8		8 9 7 3 2 11
		$\langle 3, 2, 1, 3, 1, 2 \rangle$	$\langle 3, 3, 3, 3, 0, 0 \rangle$	$\langle 2, 0, 0, 0, 4, 6 \rangle$

Pcs in bold = outer-aggregate repeated pcs

PC GRID to PROJECTION

Column 1 of PC GRID

Column 1 cross-section of PROJECTION



Pcs in bold = outer-aggregate repeated pcs

Insert outer-aggregate pcs: Simple case

- $\langle 3,2,1,3,1,2 \rangle$ = complete aggregate
- $\langle 3,3,3,3,0,0 \rangle$ = incomplete aggregate
- Missing 7 and duplicated 8
- Look to last positions of blue for missing 7 and last positions of orange for duplicated 8
- Insert 7 and push 8
- Second row – $\langle 0,2,8 \rangle$ becomes $\langle 7,0,2 \rangle$

11	4	3	5	9	10	1	8	2	0	7	6
6	7	0	2	8	1	10	9	5	3	4	11
5	6	11	1	7	0	9	8	4	2	3	10
2	9	10	8	4	3	0	5	11	1	6	7
0	5	4	6	10	11	2	9	3	1	8	7
1	8	9	7	3	2	11	4	10	0	5	6

Insert outer-aggregate pcs: Simple case

- $\langle 3,2,1,3,1,2 \rangle =$ complete aggregate
- $\langle 3,3,3,3,0,0 \rangle =$ incomplete aggregate
- Missing 7 and duplicated 8
- Look to last positions of blue for missing 7 and last positions of orange for duplicated 8
- Insert 7 and push 8
- Second row – $\langle 0,2,8 \rangle$ becomes $\langle 7,0,2 \rangle$

11	4	3	5	9	10	1	8	2	0	7	6
6	7	0	2	8	1	10	9	5	3	4	11
5	6	11	1	7	0	9	8	4	2	3	10
2	9	10	8	4	3	0	5	11	1	6	7
0	5	4	6	10	11	2	9	3	1	8	7
1	8	9	7	3	2	11	4	10	0	5	6

Insert outer-aggregate pcs: Simple

case

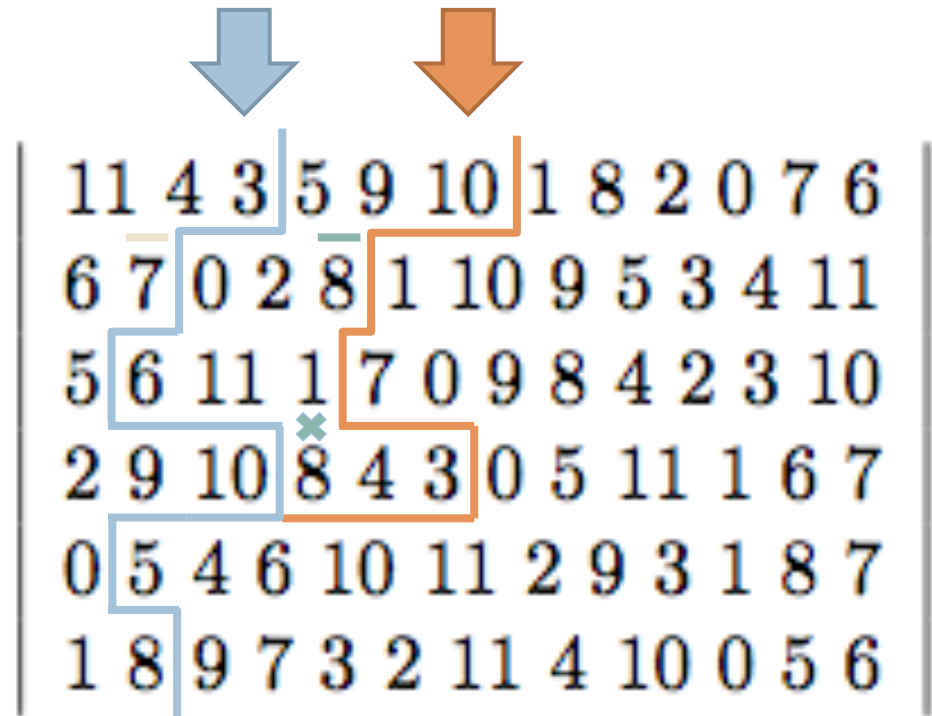
- $\langle 3,2,1,3,1,2 \rangle =$ complete aggregate
- $\langle 3,3,3,3,0,0 \rangle =$ incomplete aggregate
- Missing 7 and duplicated 8
- Look to last positions of blue for missing 7 and last positions of orange for duplicated 8
- Insert 7 and push 8
- Second row – $\langle 0,2,8 \rangle$ becomes $\langle 7,0,2 \rangle$

11	4	3	5	9	10	1	8	2	0	7	6
6	7	0	2	8	1	10	9	5	3	4	11
5	6	11	1	7	0	9	8	4	2	3	10
2	9	10	8	4	3	0	5	11	1	6	7
0	5	4	6	10	11	2	9	3	1	8	7
1	8	9	7	3	2	11	4	10	0	5	6

Insert outer-aggregate pcs: Simple

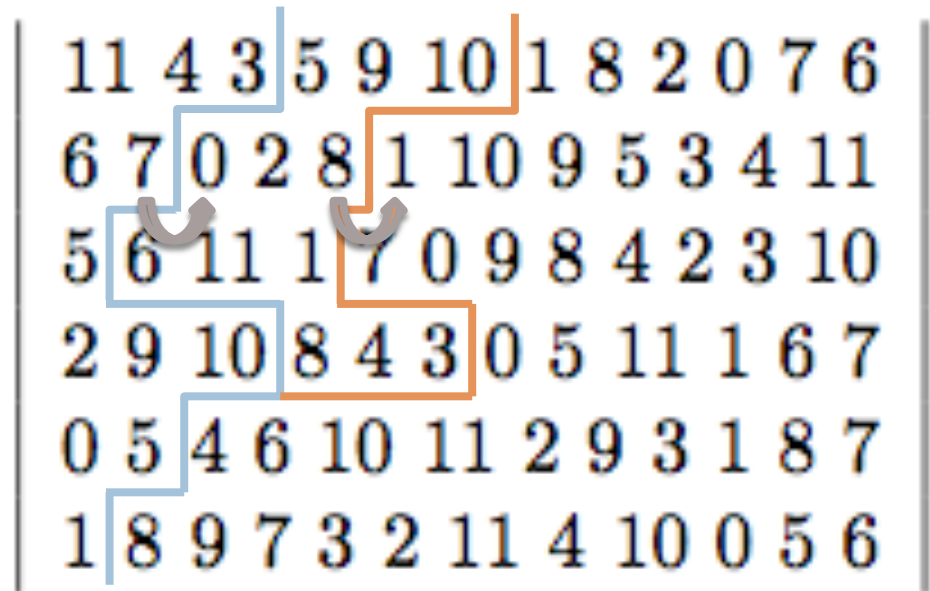
case

- $\langle 3,2,1,3,1,2 \rangle =$ complete aggregate
- $\langle 3,3,3,3,0,0 \rangle =$ incomplete aggregate
- Missing 7 and duplicated 8
- Look to last positions of blue for missing 7 and last positions of orange for duplicated 8
- Insert 7 and push 8
- Second row – $\langle 0,2,8 \rangle$ becomes $\langle 7,0,2 \rangle$



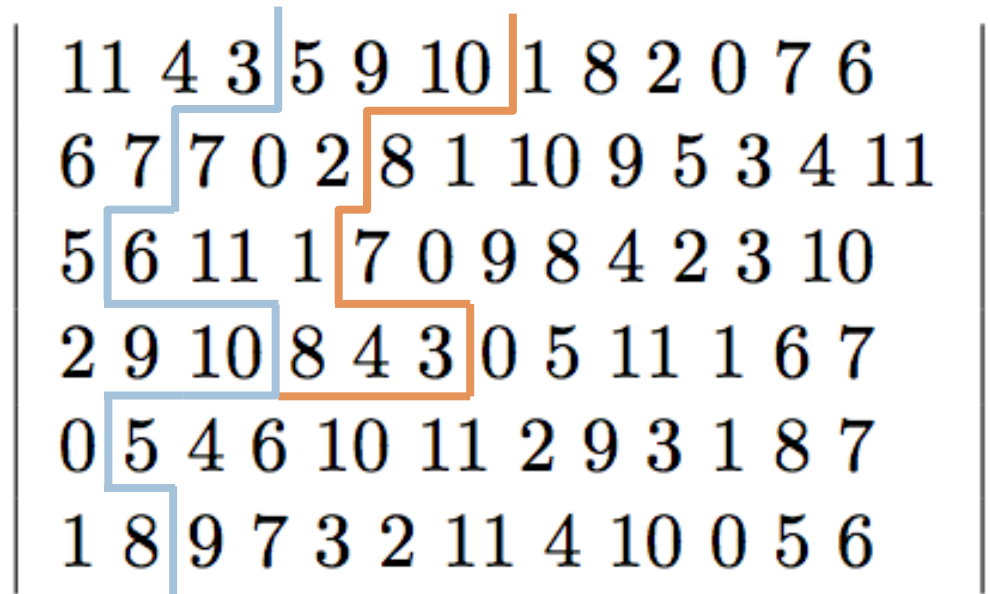
Insert outer-aggregate pcs: Simple case

- $\langle 3,2,1,3,1,2 \rangle$ = complete aggregate
- $\langle 3,3,3,3,0,0 \rangle$ = incomplete aggregate
- Missing 7 and duplicated 8
- Look to last positions of blue for missing 7 and last positions of orange for duplicated 8
- Insert 7 and push 8
- Second row – $\langle 0,2,8 \rangle$ becomes $\langle 7,0,2 \rangle$



Insert outer-aggregate pcs: Simple case

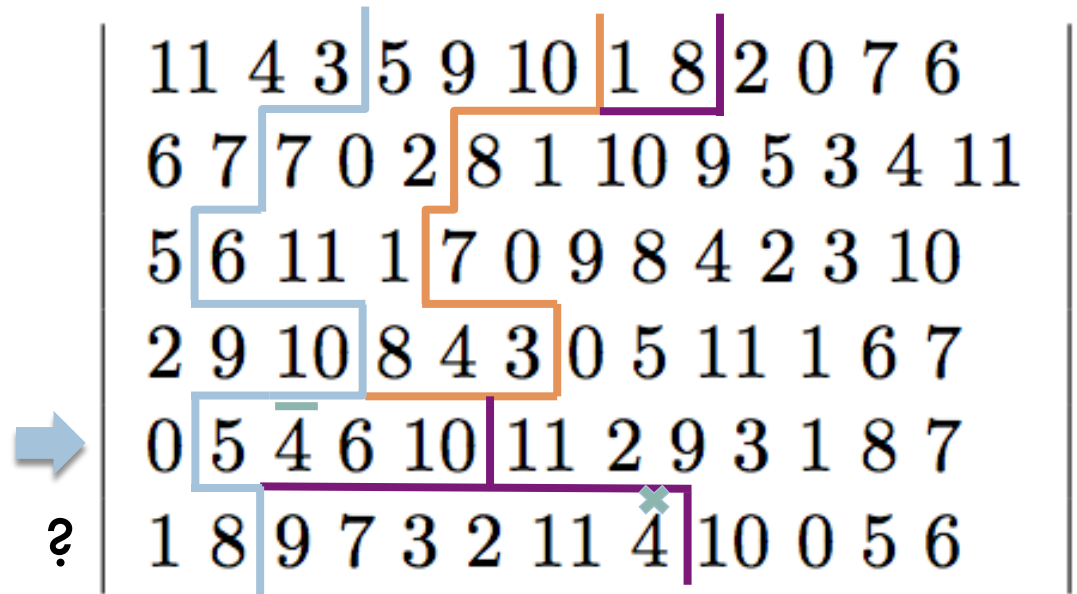
- $\langle 3,2,1,3,1,2 \rangle =$ complete aggregate
- $\langle 3,3,3,3,0,0 \rangle =$ incomplete aggregate
- Missing 7 and duplicated 8
- Look to last positions of blue for missing 7 and last positions of orange for duplicated 8
- Insert 7 and push 8
- Second row – $\langle 0,2,8 \rangle$ becomes $\langle 7,0,2 \rangle$



Complete aggregate

Insert outer-aggregate pcs: Complex case

- $\langle 2,0,0,0,4,6 \rangle =$ incomplete aggregate
- Missing 0 and duplicated 4
- Insert 0 and now missing 10
- Insert 10 and now missing 8
- Insert 8 and push duplicated 4



Insert outer-aggregate pcs: Complex case

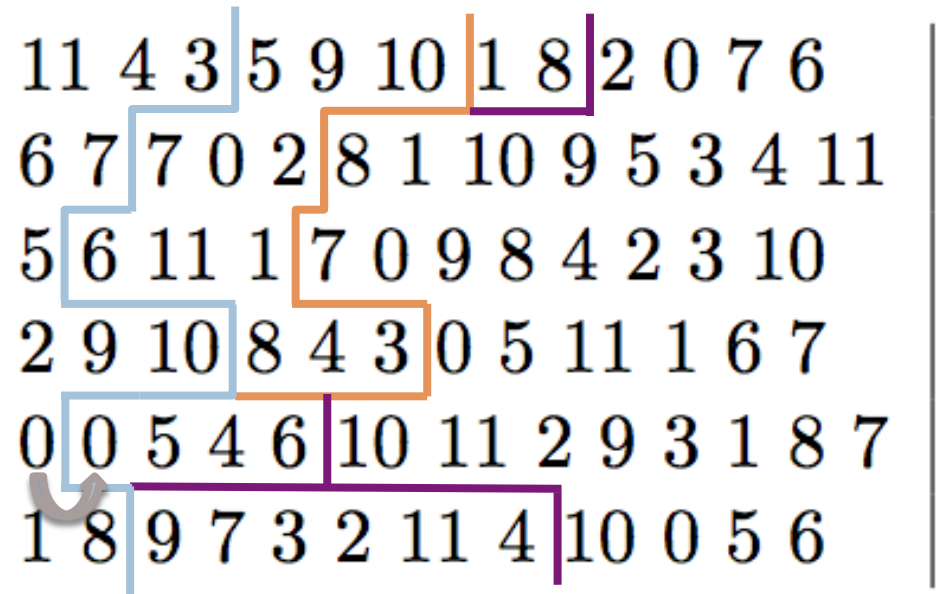
□ $\langle 2,0,0,0,4,6 \rangle =$
incomplete aggregate

□ Missing 0 and
duplicated 4

□ Insert 0 and now
missing 10

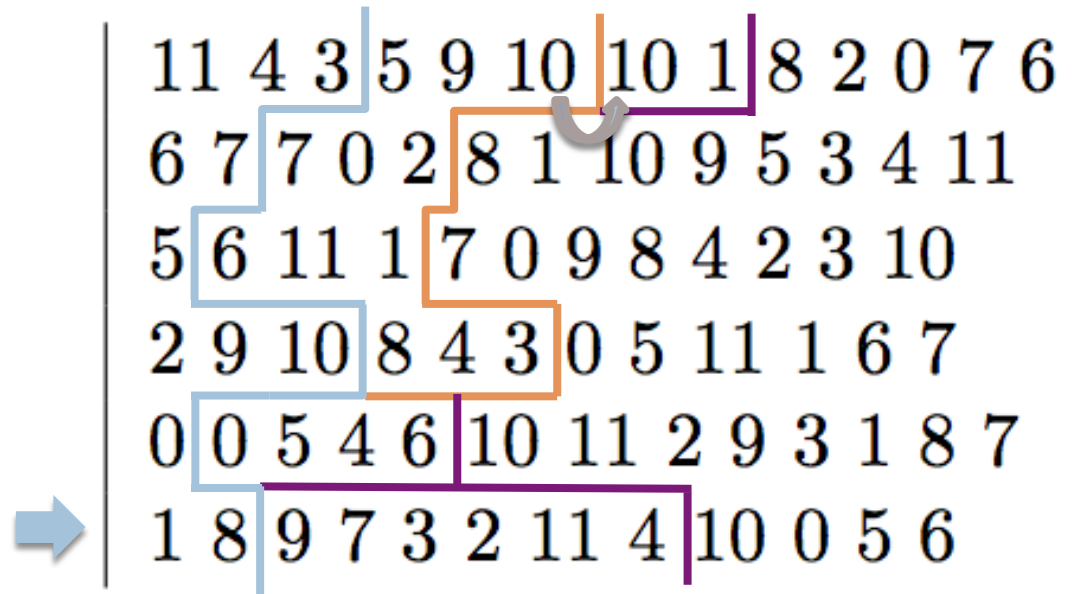
□ Insert 10 and now
missing 8

□ Insert 8 and push
duplicated 4



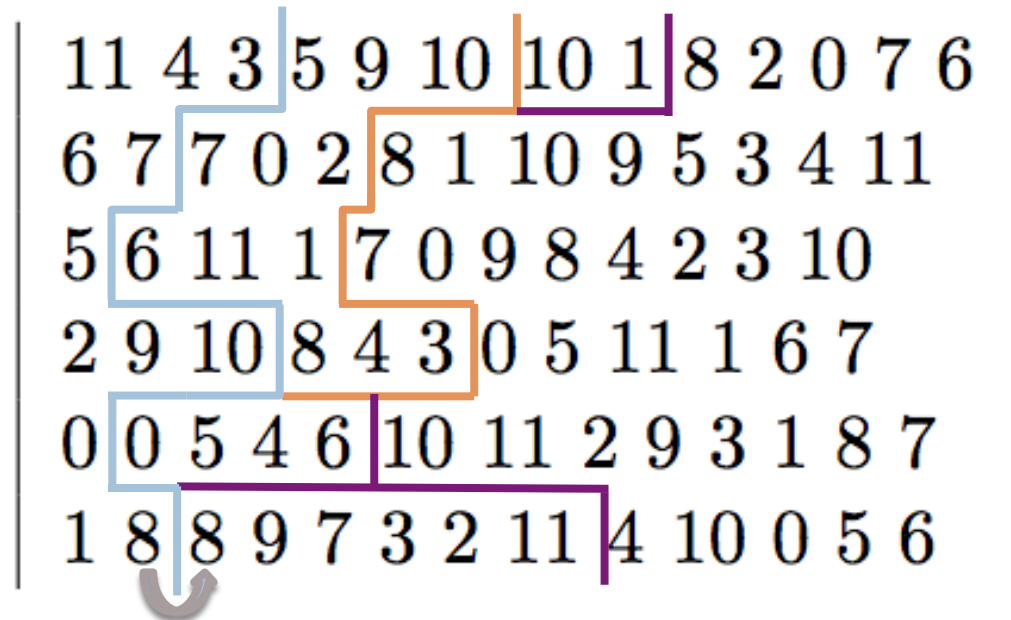
Insert outer-aggregate pcs: Complex case

- $\langle 2,0,0,0,4,6 \rangle =$ incomplete aggregate
- Missing 0 and duplicated 4
- Insert 0 and now missing 10
- Insert 10 and now missing 8
- Insert 8 and push duplicated 4



Insert outer-aggregate pcs: Complex case

- $\langle 2, 0, 0, 0, 4, 6 \rangle =$ incomplete aggregate
- Missing 0 and duplicated 4
- Insert 0 and now missing 10
- Insert 10 and now missing 8
- Insert 8 and push duplicated 4




Complete aggregate


Backtracking Babbitt algorithm

- For this partition, compute compositions that produce an aggregate with < 5 duplicates = compmat
- Select first unused. Insert pcs to complete aggregate.
- If successful, move to next partition and repeat.
- If not successful, select next unused composition and insert pcs.
- If no compositions left, backtrack and try next composition then repeat.

Partitions vector


<{3,3,2,2,1,1}, {3,3,3,3,0,0}, {6,4,2,0,0,0}, ..., >


compmat


<3,3,1,2,1,2>
<3,2,1,3,1,2>


Backtracking Babbitt algorithm

- For this partition, compute compositions that produce an aggregate with < 5 duplicates = compmat
- Select first unused. Insert pcs to complete aggregate.
- If successful, move to next partition and repeat.
- If not successful, select next unused composition and insert pcs.
- If no compositions left, backtrack and try next composition then repeat.

Partitions vector


<{3,3,2,2,1,1}, {3,3,3,3,0,0}, {6,4,2,0,0,0}, ..., >

compmat


<3,3,1,2,1,2>
<3,2,1,3,1,2>

Backtracking Babbitt algorithm

- For this partition, compute compositions that produce an aggregate with < 5 duplicates = compmat
- Select first unused. Insert pcs to complete aggregate.
- If successful, move to next partition and repeat.
- If not successful, select next unused composition and insert pcs.
- If no compositions left, backtrack and try next composition then repeat.

Partitions vector

\downarrow
 $\langle \{3,3,2,2,1,1\}, \{3,3,3,3,0,0\}, \{6,4,2,0,0,0\}, \dots, \rangle$

compmat



$\langle 3,3,1,2,1,2 \rangle$

$\langle 3,2,1,3,1,2 \rangle$

compmat empty

Backtracking Babbitt algorithm

- For this partition, compute compositions that produce an aggregate with < 5 duplicates = compmat
- ✕ □ Select first unused. Insert pcs to complete aggregate.
- If successful, move to next partition and repeat.
- If not successful, select next unused composition and insert pcs.
- If no compositions left, backtrack and try next composition then repeat.

Partitions vector

$\langle \{3,3,2,2,1,1\}, \{3,3,3,3,0,0\}, \{6,4,2,0,0,0\}, \dots, \rangle$

compmat $\left\{ \begin{array}{l} \langle 3,3,1,2,1,2 \rangle \\ \langle 3,2,1,3,1,2 \rangle \end{array} \right.$ compmat empty

Backtracking Babbitt algorithm

- For this partition, compute compositions that produce an aggregate with < 5 duplicates = compmat
- Select first unused. Insert pcs to complete aggregate.
- If successful, move to next partition and repeat.
- If not successful, select next unused composition and insert pcs.
- If no compositions left, backtrack and try next composition then repeat.

Partitions vector

\downarrow
 $\langle \{3,3,2,2,1,1\}, \{3,3,3,3,0,0\}, \{6,4,2,0,0,0\}, \dots, \rangle$

compmat

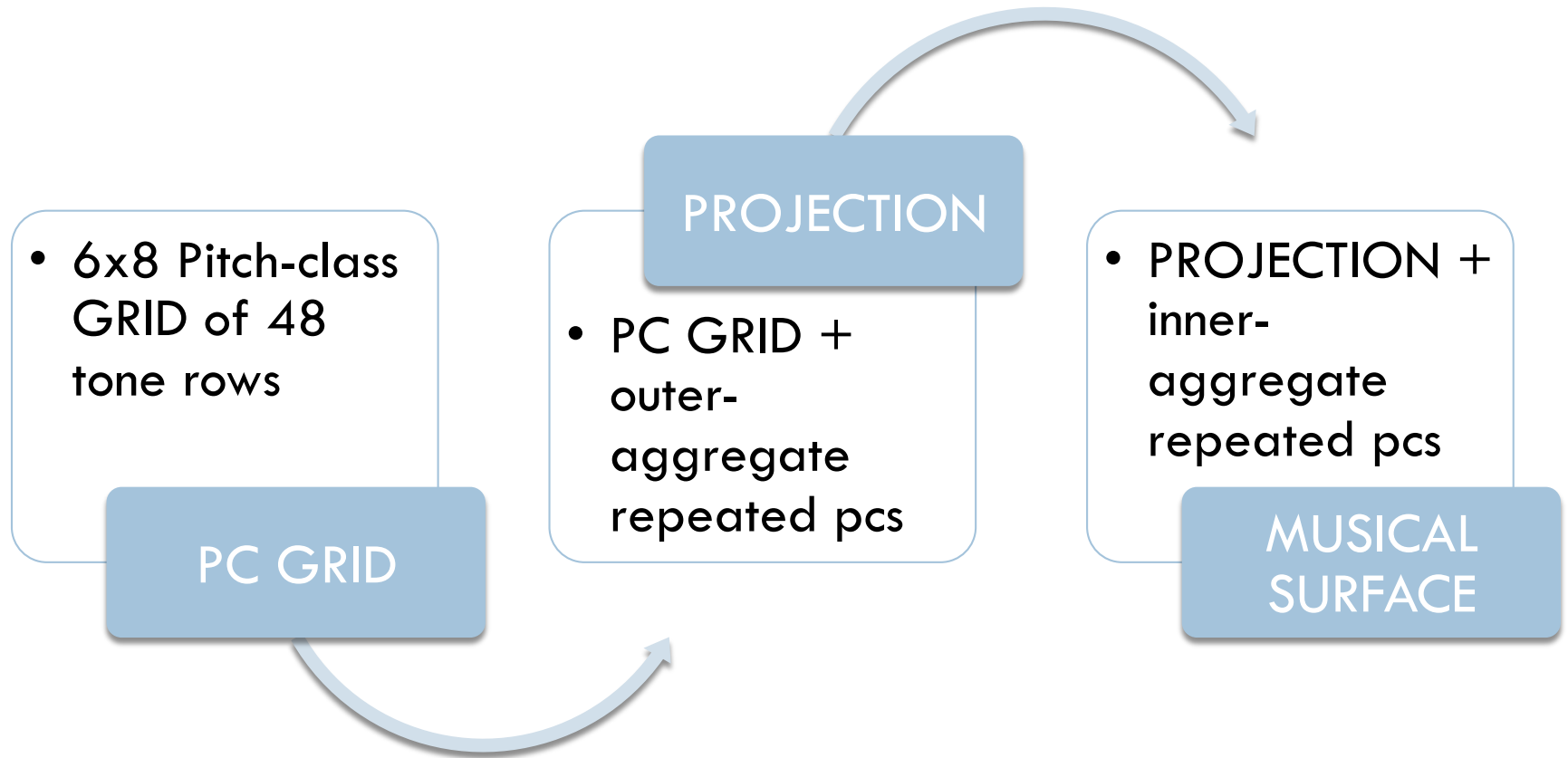
$\langle 3,3,1,2,1,2 \rangle$

$\langle 3,2,1,3,1,2 \rangle$

What might these compositions sound like? $\langle 2,2,2,2,2,2 \rangle$ | $\langle 6,6,0,0,0,0 \rangle$

The image displays a musical score for guitar, consisting of three staves of music. The top staff shows a sequence of notes with a five-fingered scale (5 = d) and a section marked 'sul pont. ord.' (sul ponticello) with a dynamic marking of 'ppp'. The middle staff begins at measure 92, featuring a triplet (3 = d) and a seven-fingered scale (7 = d) with a dynamic marking of 'ff'. The bottom staff begins at measure 96, showing a triplet (3 = d) and various dynamic markings including 'f', 'ff', and '< ff', along with accents (> f). The score includes various musical notations such as slurs, ties, and dynamic markings.

Compositional Process



58% of pcs accounted for

Future research