a) Erklären Sie die Abkürzung SMP und NUMA.

Um die Speicherbandbreite zu erhöhen kann auf **SMP**-Systeme gesetzte werden. Hierbei werden die Speicher-Controller in die CPU integriert, sodass der Arbeitsspeicher direkt mit der CPU verbunden werden kann. Es wird also weder ein externer Speicher-Controller noch die Northbridge dazu verwendet um mit dem Arbeitsspeicher zu kommunizieren.
Bei einem SMP-System steht jedem Prozessor eine Speicherbank zu Verfügung (Local Memory). So kann z.B. bei einem System mit zwei Prozessoren die Speicherbandbreite verdoppelt werden ohne dass z.B. die Bandbreite der Northbridge erhöht werden muss.
Da jedoch nach wie vor alle Prozessoren auf alle Speicherbänke zugreifen können müssen, ergibt sich der Nachteil von NUMA (Non-Uniform Memory Architecture). Will z.B. die CPU_1 auf den Speicher von CPU_2 zugreifen, so muss erst die Verbindung von CPU_1 zu CPU_2 überbrückt werden. Der Vorgang dauert also länger als wenn CPU_1 auf ihren lokalen Speicher zugreifen möchte. Bei mehr als zwei CPUs verlangsamen weitere Verbindungen zwischen den Prozessoren den Speicherzugriff. Die zusätzliche Zeit, die für den Zugriff über solche Verbindungen in Anspruch nimmt, wird mit dem sogenannten NUMA-Faktor angegeben.
Muss ein Programm für ein SMP-System mit NUMA entwickelt, muss dieses Problem beachtet werden.

b) Was ist die *von Neumann* Architektur? Was kostet ein Zugriff auf ein Register bzw. auf den Hauptspeicher? Was versteht man unter *eviction*?

Bei der von Neumann Architektur wird pro Prozessorkern ein gemeinsamer Cache sowohl für Programmcode als auch für Daten verwendet. Im Gegensatz dazu wird z.B. bei Intel-Systemen sowohl ein Cache für den Programmcode als auch ein Cache für die Daten zur Verfügung gestellt.
Der Zugriff auf ein Register benötigt maximal einen Prozessorzyklus, während der Zugriff auf den Arbeitsspeicher mehrere 100 Prozessorzyklen (Pentium M: ca. 240) benötigt. Darum gibt es schnelle Caches die dazwischengeschaltet sind, um Daten für die spätere Verarbeitung im Arbeitsspeicher vorzubereiten.
Unter eviction versteht man die Räumung bzw. Freimachung von Speicherplatz, wenn Daten von einem Cache (z.B. L1d) in einen anderen Cache (z.B. L2) transportiert werden. Dabei kann man zwei

verschiedene Arten von Caches unterscheiden: exklusive und inklusive Caches. Bei exklusiven Caches funktioniert eviction wie oben beschrieben. Im Gegensatz dazu wird bei inklusiven Caches eine Cache-Reihe sowohl in L1d als auch in L2 gespeichert. Darum geht hier das freimachen von Speicher (von L1d auf L2) schneller.

c) Wofür braucht man virtuellen Speicher? Warum wird mehrstufige *address translation* verwendet?

Virtueller Speicher kann einem bestimmten Prozess vom Betriebssystem zur Verfügung gestellt werden. Der Adressraum des virtuellen Speichers ist vom tatsächlichen Arbeitsspeicher des Systems unabhängig.
Die Umsetzung von realen in virtuelle Spicheradressen übernimmt die MMU (memory management unit), welche heutzutage zusammen mit der CPU auf einem Chip untergebracht ist.
Einem Prozess wird so vorgegaukelt, dass er alleine auf den gesamten Speicher Zugriff hat. Benötigt ein Prozess mehr Speicher als tatsächlich an Arbeitsspeicher zur Verfügung steht so kann dieser auf die Festplatte ausgelagert werden.
Werden Einträge benötigt, die sich nicht im Arbeitsspeicher befinden so werden diese von der Festplatte in den Arbeitsspeicher kopiert und können so genutzt werden.

Das Problem von einstufiger Adressumsetzung ist, dass eine Seitentabelle bei einem großen virtuellen Adressraum und/oder kleiner Seitengröße sehr viel Speicher benötigt (Seitentabellen werden im Arbeitsspeicher gespeichert). Dazu kommt, dass jeder Prozess eine eigene Seitentabelle benötigt. Das heißt ein Großteil des realen Speichers wird alleine für die Adressumwandlung (für die Seitentabellen) verschwendet.
Bei mehrstufiger Adressumsetzung werden mehrere Stufen von Seitentabellen durchlaufen. Dadurch können die einzelnen Seitentabellen klein gehalten werden und benötigen somit auch weniger Speicher.

d) Was versteh man unter *locality*? Was versteht man unter *alignment*?

Locality bedeutet dass Befehle bzw. Daten entweder oft hintereinander ausgeführt werden (spatial locality) oder aber auch dass diese oft im Adressraum direkt hintereinander vorkommen (temporal locality). Diesen Umstand machen sich Caches zu Nutze.

Um einen schnellen Speicherzugriff auf ein Datenelement zu gewährleisten, muss dessen Speicheradresse ein ganzzahliges Vielfaches eines Chunks sein. Ein Chunk entspricht dabei der Größe eines Wortes (auf einem 32 Bit System: 4 Byte). Ist ein Datenelement z.B. 6 Byte groß, so sollte es so ausgerichtet sein wie ein Datenelement mit 8 Byte (2 x 4 Byte). Ist das der Fall dann ist es ausgerichtet (aligned). Das kann in diesem Beispiel dadurch erreicht werden, dass die übrigbleibenden 2 Bytes aufgefüllt werden.